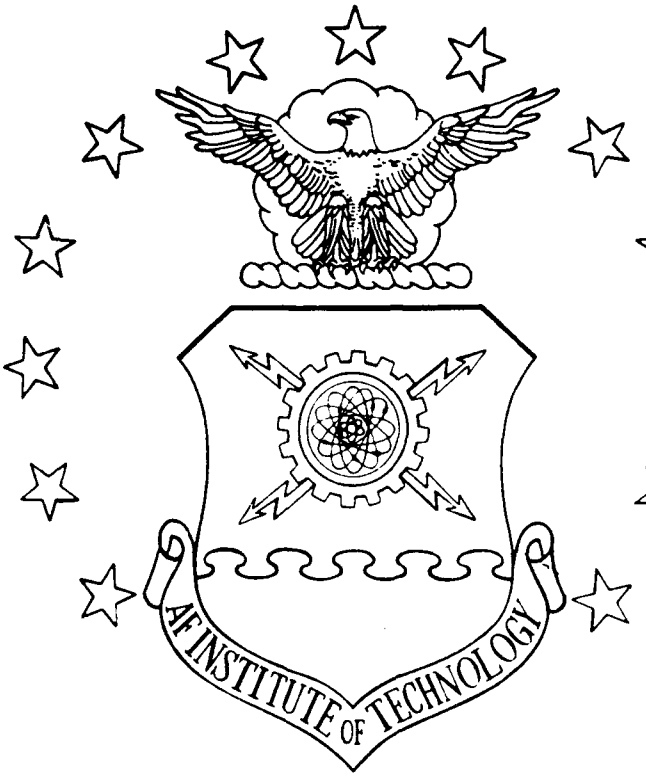





# RELATIVE UTILITY OF SELECTED SOFTWARE REQUIREMENT METRICS

THESIS

James H. Byers, Captain, USAF

AFIT/GSS/LSY/91D-4





DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSS/LSY/91D-4

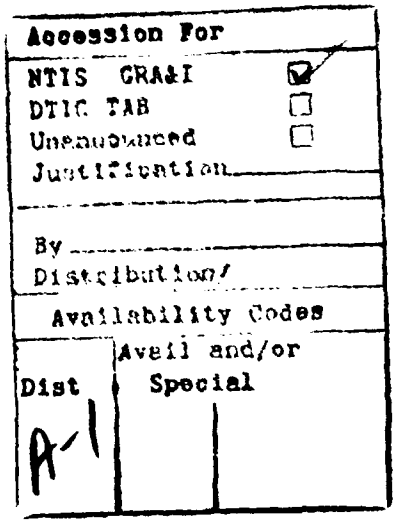
# RELATIVE UTILITY OF SELECTED SOFTWARE REQUIREMENT METRICS

THESIS

James H. Byers, Captain, USAF

AFIT/GSS/LSY/91D-4

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.



AFIT/GSS/LSY/91D-4

# RELATIVE UTILITY OF SELECTED SOFTWARE REQUIREMENT METRICS

THESIS

Presented to the Faculty of the School of Systems and Logistics
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Software Systems Management

James H. Byers, B.S.
Captain, USAF

December 1991

## Preface

The objective of this study was to determine the relative utility of selected software requirement metrics in assessing the productivity of the software requirements analysis process and the quality of the products of this process. A broader goal was to provide some of the information needed to make intelligent choices of requirement metrics on which to focus further research efforts and, more importantly, to use on future software developments. This objective was met by collecting information about the perceptions that practicing software professionals have of the usefulness of selected requirement metrics.

Many people have played a role in conducting this study. My thanks extend to the various people who took the time to respond to my rather lengthy questionnaire. Thanks also go to my thesis advisor, Mr Dan Ferens, for providing the guidance and freedom that allowed me to complete this study. Finally, I am especially grateful to my wonderful wife Lauren for helping me keep this research effort in the proper perspective.

James H. Byers

## Table of Contents

## Table of Contents (cont'd)

## List of Figures

## List of Tables

## Abstract

The objective of this study was to determine the relative utility of selected software requirement metrics in assessing the productivity of the software requirements analysis process and the quality of the products of this process. This objective was met by collecting information about the perceptions that practicing software professionals have of the usefulness of various requirement metrics.

The study employed a two part methodology. The first part utilized Basili's goal/question/metric paradigm to identify specific goals of the measurement effort and to identify requirement metrics worthy of further investigation. The second part employed a typical research design to gather perceptions that software professionals have of the utility of several metrics selected from those identified earlier.

The study produced inconclusive results and further research is recommended. Results were based on a small sample and the data only reiterated the mixed opinions that software professionals have of the usefulness of software metrics. One significant finding is the consensus that a metric must be precisely defined for it to be accepted by the software community.

# RELATIVE UTILITY OF SELECTED SOFTWARE REQUIREMENT METRICS

## I. Introduction

### General Issue

Managing software development has become increasingly difficult as the software programs under development have become more complex and the number of tasks that software programs perform has grown. Because software development has grown more complicated, the costs and length of development efforts have rapidly increased and continue to rise. As developments have grown more complex and costly, an axiom has emerged from the software development field: The proper use of software metrics is essential to the successful management of software development efforts (Mills, 1988:15). Furthermore, the use of software metrics shows great promise in enhancing the quality of software products. As software quality has become an increasingly important issue, measuring

the quality of software processes and products has also increased in importance.

Software metrics are used to quantitatively measure the essential features of software so that comparison of the features can be made against some standard; usually a requirement, goal, or expectation. Software metrics are classified as either process or product metrics and are applied to either the development process or the software product developed. Process metrics quantify attributes of the development process and environment, whereas product metrics measure characteristics of the software product.

Several recent reports have highlighted the usefulness of software metrics as tools used in software developments. The 1989 National Research Council Air Force Studies Board Committee on Adapting Software Development Policies to Modern Technology recommended the Air Force mandate the use of software engineering environments of which the application of software metrics is a vital part. The committee also specifically recommended the Air Force use software metrics as quality indicators and to enhance evaluation of software characteristics. A 1987 Defense Science Board report on military software included recommendations that the Department of Defense develop software metrics to measure software quality, completeness, and implementation progress. Lastly, a 1984 Air Force Studies Board also recommended the

Air Force develop tools, including software metrics, to aid in software developments. (National Research Council, 1989:28, 38, 47–50, 63–66)

## Specific Problem

Software metrics have been developed for and are used during virtually every phase of the software life cycle. The first metrics were developed to measure source code program length and complexity. Efforts were then made to correlate these measurements with development and maintenance costs and efforts. More recently, metrics which quantify the qualitative attributes of software designs have gained prominence. However, relatively few metrics have been developed for use during the requirements analysis phase of the software life cycle. This is truly unfortunate, since requirement metrics can be particularly useful.

Requirement metrics serve many purposes. Requirement metrics can be used for project cost estimation and manpower allocation. They can be used to assess and reduce the complexity of the requirement specification by identifying inconsistent or poorly structured requirements. Requirement metrics can be more useful than other metrics; they can be used to reduce the complexity of the design process, and to make intelligent tradeoffs between manpower allocations among

projects, project deadlines, and software performance targets. They can help in choosing between alternative strategies for later phases of the development life cycle; for instance, to guide design and testing. They can also help in deciding if and how to use complexity reduction techniques. In summary, requirement metrics can be useful because they are applied in the earliest stage of development and can help guide analysts and developers to better results during system development. (Ramamoorthy, 1986:81; Ramamoorthy, 1985:111-112)

In order to make intelligent choices of requirement metrics on which to focus further research efforts and to use on future software developments, information about the utility of various metrics is needed. However, only a meager amount of research has been conducted to identify useful requirement metrics; i.e., requirement metrics that can be used effectively. This research effort is intended to provide some of this information.

## Research Objectives

The objective of this study is to determine the relative utility of selected software requirement metrics in assessing the productivity of the requirements analysis process and the quality of the products of this process. This objective is

met by determining the relative utility of selected software requirement metrics to practicing software engineers and computer scientists, persons serving in a software engineer's or computer scientist's position, software systems project managers, persons teaching courses in a software engineering or computer science curriculum, and persons performing research in the area of software metrics. (Hereafter, these persons are referred to as software professionals.) The perceptions these software professionals have of the utility of software requirement metrics provide an indication of the usefulness of the metrics when applied to actual requirements analyses.

## Research Scope

The first part of this study includes an extensive review of past research and available documentation on various topics dealing with requirements analysis and software metrics. The second part is limited to an analysis of the perceptions that software professionals have of the utility of selected requirement metrics in assessing the productivity of the requirements analysis process and the quality of the products produced by this process.

This study does not address the correlation of requirement metric measures, estimates, and predictions with

actual software product or process characteristics. Furthermore, no attempt is made to collect data about quantifiable attributes of the software requirement process or products, nor is any attempt made to measure the competency of the software professionals taking part in this study. Rather, this study focuses on the utility of selected or candidate metrics for use during software requirements analyses.

## Basic Methodology

Although a detailed discussion of the research methodology is provided in Chapter III, an overview is provided here to help the reader better understand the ensuing text.

The design of this study is based in part on the goal/question/metric paradigm proposed by Basili and Rombach. This paradigm consists of three steps. The first step is to identify a goal or set of goals. In most cases the goal is to improve a process or product. The second step involves formalizing the goal into questions that, if answered, result in realization of the goal. The questions must focus on factors that are germane to the goal and are usually posed in the form: Does a relationship exist between input variable A and output variable B? In the third and last step, metrics

are developed or identified to provide the information needed to answer the questions posed in step two. (Basili, 1987:350–351; Shepperd, 1990:312)

After these actions have been performed, the metrics identified in step three must be examined to see which will most likely help attain the goals set in step one. This may be accomplished in several ways. The method used in this effort was to survey software professionals about their perceptions of the utility of the metrics selected in step three.

## Investigative Questions

In order to determine the relative utility of various requirement metrics, several issues must be investigated. The first of these issues are posed in the form of questions using the goal/question/metric paradigm:

(1) What are the specific goals for improving the requirements analysis phase of the software development life cycle?

(2) What questions can quantify these goals?

(3) What metrics might provide the information needed to answer these questions?

Once these questions have been answered, specific questions can be asked about the metrics identified in the answer to question three:

(4) What is the relative ranking, in terms of perceived utility, of the software metrics available for use during requirements analysis?

(5) Which requirement metrics, if any, are ranked significantly higher than the others? Which are ranked significantly lower?

(6) Are there significant differences between the rankings for product and process type metrics?

(7) Is there a significant correlation between a software professional's experience and the perception of the utility of a particular requirement metric?

## Background

As stated earlier, software metrics are used to quantitatively measure the essential features of software so that comparison of the features can be made against some standard; usually a requirement, goal, or expectation. Although metrics have been developed for and used during

virtually every phase of the software life cycle, and even though requirement metrics can be more useful than other metrics, relatively few requirement metrics have been developed. A discussion of these requirement metrics is provided in the following chapter.

## II. Literature Review

This chapter is intended to provide a review of the software metrics that have been developed for use during the requirements analysis phase of the software life cycle. To this end, this chapter is essentially a compendium of various requirement metrics. Additionally, other measures and measurement techniques which fall under a loose definition of the term "software metric" and which may be of value during requirements analysis are discussed. For organizational purposes, this literature review is divided into two sections corresponding to the two basic categories of metrics; process and product metrics. Product metrics have been further categorized according to the characteristic of the product they are intended to measure. Lastly, please note that the terms "metric" and "measure" used throughout the following text are interchangeable.

### Process Metrics

The IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software states that "process measures address cause and effect of both the static and dynamic aspects of the development and support management processes necessary for maximizing productivity and quality" (IEEE, 1989:27). In other words, process metrics are

measures applied to the development process in order to quantify attributes of the development process and environment (Conte, 1986:19–20). These measurements are made in order to better understand the process and, eventually, improve it.

The IEEE Standard Dictionary of Measures to Produce Reliable Software identifies several metrics that may be applied to the requirements analysis process. The first measure included in this discussion, named the fault-days number, "represents the number of days that faults spend in the software system from their creation to their removal." The information collected for this measure includes the phase and date when the fault was introduced in the system, and the phase, date, and time when the fault is removed. The fault-days number is computed as follows:

> For each fault detected and removed, during any phase, the number of days from its creation to its removal is determined (fault-days).
>
> The fault-days are then summed for all faults detected and removed, to get the fault-days number at system level, including all faults detected/removed up to the delivery date. In cases when the creation date is not known, the fault is assumed to have been created at middle of the phase in which it was introduced. (IEEE, 1989:42–43; IEEE, 1988:16–17)

The second applicable process measure from the IEEE Standard is the error distribution measure. This metric "involves the analysis of the defect data collected during

each phase of the software development" and "allows ranking of the predominant failure modes." The data needed to perform this measure is collected in order to adequately describe the errors. This data includes error type, error severity, the phase the error was introduced into the system, and measures that can be taken to prevent reoccurrence of similar errors. Faults associated with one another are identified, as is each fault's discovery mechanism, including the reasons any associated faults were previously undetected. The error distribution is determined as follows:

> The [data] for each error are recorded and the errors are counted according to the criteria adopted for each classification. The number of errors are then plotted for each class. The errors are classified and counted by phase, by the cause, and by the cause for deferred fault detection. Other similar classifications could be used such as the types of steps suggested to prevent the reoccurrence of similar errors or the types of steps suggested for earlier detection of the corresponding faults. (IEEE, 1989:49–51; IEEE, 1988:19)

The IEEE Standard also describes a measure of the manhours per major defect detected during software product inspections. "This measure provides a quantitative figure that can be used evaluate the efficiency of the ... inspection process" and can be used to evaluate inspections of requirements specifications. The data needed to compute this measure are:

> $T_1$ = Time expended by the inspection team in preparation for the inspection meeting.

$T_2$ = Time expended by the inspection team in conduct of an inspection meeting.

$S_i$ = Number of major (non-trivial) defects detected during the $i^{th}$ inspection.

I = Total number of inspections to date.

This measure is implemented as follows:

At each inspection meeting, record the total preparation time expended by the inspection team. Also, record the total time expended in conducting the inspection meeting. All defects are recorded and grouped into major/minor categories. (A major defect is one which must be corrected for the product to function within specified requirements.)

The inspection times are summarized and the defects cumulatively added.

The manhours per major defect detected is:

$$M = \frac{\sum_{i=1}^{I} (T_1 + T_2)_i}{\sum_{i=1}^{I} S_i}$$

(IEEE, 1989:52–53; IEEE, 1988:19)

The last applicable process metric identified in the IEEE Standard, named the defect index, is actually both a process and a product measure. "This measure provides a continuing, relative index of how correct the software is as it proceeds through the development cycle." It consists of eight primitives for each life cycle phase, including three weighting factors for defect severity level. (An item is considered primitive if it cannot be partitioned into

subordinate components. For the purposes of this study, a primitive is an elementary data item.) The primitives from each phase are mathematically combined to compute a phase index, after which all phase indices are summed to compute the overall defect index. (IEEE, 1989:48–49; IEEE, 1988:18–19)

The last process measure which may be used during requirements analysis and, in fact, during any and all life cycle phases is a schedule progress metric proposed by Schultz. This measure "tracks the … ability to maintain the software development schedule by tracking the delivery of software work packages defined in the work breakdown structure." It takes the following form:

$$\text{Estimated Schedule (months)} = \frac{\text{Program Schedule (months)}}{\frac{\text{BCWP}}{\text{BCWS}}}$$

where BCWP is the budgeted cost of work performed and BCWS is the budgeted cost of work scheduled; two fairly common cost accounting terms. (Schultz, 1988:22)

## Product Metrics

Product metrics are measures applied to a software product in order to quantify attributes of that product (Conte, 1986:19–20). These measurements are made in order to better understand and, eventually, improve the product. The

product to be measured and of interest in this research effort is the product of the requirements analysis process; a formal or informal software requirements specification.

For organizational purposes, it is convenient to associate possible requirement product metrics with the qualities they are intended to measure. To that end, the most appropriate qualities to use are the characteristics of good software requirements specifications (SRS) as defined in the IEEE Guide to Software Requirements Specifications. According to the IEEE Guide, a good SRS is (1) unambiguous, (2) complete, (3) verifiable, (4) consistent, (5) modifiable, (6) traceable, and (7) useable during the operation and maintenance phase. (IEEE, 1984:11-13)

Measures of Requirement Ambiguity. (IEEE characteristic (1) unambiguous.) Gause and Weinberg have proposed using "an ambiguity poll to estimate the ambiguity of a requirement." This ambiguity poll is performed "whenever a piece of requirements work is said to be finished" with the purpose of identifying ambiguous requirements. The poll is conducted as follows:

> (1) Gather a group of people to answer questions about the document whose ambiguity is to be measured.
>
> (2) Be sure that there is no pressure to conform, or no influence of any sort of one participant on another.

(3) Propose a set of questions, each of which can be answered with a number, such as: How fast? How big? How expensive? What capacity?

(4) Estimate the ambiguity by comparing the highest and lowest answers.

(5) Interview the high and low estimators to help locate the sources of the ambiguity.

Gause and Weinberg advise that, to obtain the most reliable results, "the group used for estimating ambiguity should be as diverse as possible, at the very least including a sample from each population that will be affected by the eventual product." (Gause, 1989:217–224)

Cioch proposes a similar technique to measure an individual's misinterpretation and comprehension of statements. His method, which may also be used to identify ambiguous requirements, is based on short-answer questions as opposed to the open-ended questions used by Gause and Weinberg. Cioch describes his technique as follows:

> The proposed approach to measuring misinterpretation and comprehension involves the use of short-answer items in a test instrument. In order to differentiate between misinterpretation and comprehension, the measurement technique must be able to distinguish between not knowing the correct answer and giving a wrong answer.
>
> The short-answer question type developed to yield this distinction is a modified version of a standard true/false question. ... [The] participant is presented with a collection of statements, the truth or falsity of which must be judged. Instead of responding true or false, the participant answers with a number from 1 to 5, depending upon which of the following best describes the participant's view of the statement's veracity:

1 = I am certain this statement is false
2 = I am fairly sure this statement is false
3 = I don't know
4 = I am fairly sure this statement is true
5 = I am certain this statement is true

[Points are awarded based on whether the statement is, in actuality, true or false, and on the participant's response.] ... For the measure of comprehension, points are awarded only when the participant is either fairly sure or certain of the correct answer. ... In measuring misinterpretation, points are awarded only when the participant is either fairly sure or certain of an answer, and that answer is incorrect. ... Assuming the particular statement is true, the following relationship exists between points awarded for the comprehension and misinterpretation measures:

**Comprehension and Misinterpretation Measurement Scales**

| Response | Comprehension | Misinterpretation |
|---|---|---|
| I am certain this statement is false | 0 | 2 |
| I am fairly sure this statement is false | 0 | 1 |
| I don't know | 0 | 0 |
| I am fairly sure this statement is true | 1 | 0 |
| I am certain this statement is true | 2 | 0 |

Figure 1. Comprehension and Misinterpretation Measurement Scales (Cioch, 1991:87)

A complementary measurement scheme is used when the statement is false. (Cioch, 1991:86-87)

The IEEE Standard Dictionary of Measures to Produce Reliable Software identifies the technique of cause and effect graphing as a means of identifying both ambiguous and incomplete requirements:

> Cause and effect graphing aids in identifying requirements that are incomplete and ambiguous.

This [technique] explores the inputs and expected outputs of a program and identifies the ambiguities. Once these ambiguities are eliminated, the specifications are considered complete and consistent.

[Furthermore,] a cause and effect graph is a formal transformation of a natural language specification [for example, written in English] into its input conditions and expected outputs.

The primitives (data) needed to compute this measure include:

List of causes: distinct input conditions

List of effects: distinct output conditions or system transformation (effects are caused by the changes in the state of the system)

$A_{existing}$ = number of ambiguities in a program remaining to be eliminated

$A_{tot}$ = total number of ambiguities identified

The measure is computed as follows:

Identify all requirements and divide them into separate entities. Analyze the requirements to identify all the causes and effects in the specification. After the analysis is completed, assign each cause and effect a unique identifier. For example, E1 for effect one or I1 for input one.

To create the cause and effect graph:

(1) Represent each cause and each effect by a node identified by its unique number.

(2) Interconnect the cause and effect nodes by analyzing the semantic content of the specification and transforming it into a Boolean graph. Each cause and effect can be in one of two states: true or false. Using Boolean logic, set the possible states of the causes and determine under what conditions each effect will be present.

(3) Annotate the graph with constraints describing combinations of causes and effects that are impossible because of semantic or environmental constraints.

(4) Identify as an ambiguity any cause that does not result in a corresponding effect, any effect that does not originate with a cause as a source, and any combination of causes and effects that are inconsistent with the requirement specification or impossible to achieve.

The measure [of ambiguities present] is computed as follows:

$$CE(\%) = 100 \times (1 - \frac{A_{existing}}{A_{tot}})$$

When all of the causes and effects are represented in the graph and no ambiguities exist, the measure is 100%. A measure of less than 100% indicates some ambiguities still exist. (IEEE, 1989:45–47; IEEE, 1988:17–18)

The IEEE Standard describes a second graphical technique that can be used to identify requirements which may be misinterpreted. This technique and its associated measures of requirements compliance are ascertained through a graphical analysis of the software requirements specification. This metric, appropriately named the requirements compliance measure, can be used to identify and quantify inconsistencies, incompleteness, and misinterpretations in the software requirements specification.

This analysis is used to verify requirements compliance by using system verification diagrams (SVDs), a logical interconnection of stimulus response elements, (e.g., stimulus and response) that detect inconsistencies, incompleteness, and misinterpretations.

The primitives for this measure are:

Decomposition elements (DEs):

Stimulus = external input
Function = defined input/output process
Response = result of the function
Label = numerical DE identifier
Reference = specification paragraph number

Requirement errors detected using SVDs:

$N_1$ = Number due to inconsistencies
$N_2$ = Number due to incompleteness
$N_3$ = Number due to misinterpretation

The implementation of an SVD is composed of the following phases:

(1) The decomposition phase is initiated by mapping the system requirement specifications into stimulus/response elements (DEs). That is, all keywords, phrases, functional and/or performance requirements and expected outputs are documented on decomposition forms.

(2) The graph phase uses the DEs from the decomposition phase and logically connects them to form the SVD graph.

(3) The analysis phase examines the SVD from the graph phase by using connectivity and reachability matrices. The various requirement error types are determined by examining the SVD and identifying errors as follows:

(a) Inconsistencies – Decomposition elements that do not accurately reflect the system requirement specification.

(b) Incompleteness – Decomposition elements that do not completely reflect the system requirement specification.

(c) Misinterpretation – Decomposition elements that do not correctly reflect the system requirement specification. These errors may occur during translation of the requirements into decomposition elements, constructing the SVD graph, or interpreting the connectivity and reachability matrices.

An analysis is also made of the percentages for the various requirements error types for the respective categories: inconsistencies, incompleteness, and misinterpretation.

$$\text{Inconsistencies} \quad (\%) = \frac{N_1}{(N_1 + N_2 + N_3)} \times 100$$

$$\text{Incompleteness} \quad (\%) = \frac{N_2}{(N_1 + N_2 + N_3)} \times 100$$

$$\text{Misinterpretation} \quad (\%) = \frac{N_3}{(N_1 + N_2 + N_3)} \times 100$$

(IEEE, 1989:70–72; IEEE, 1988:25–26)

The next measure is not a measure of the ambiguity of requirement statements, rather it is a measure of the understandability of semantic statements; i.e., written in English. However, this measure, the Flesch-Kincaid readability formula, may be useful in identifying poorly stated requirements such as easily misunderstood or ambiguous requirements. The measure is also very simple, and is described here:

> The formula has two factors: (1) sentence length in words and (2) word length in syllables. It provides grade level (GL) according to the formula:

$$GL = 0.39 \text{ (Average number of words per sentence)} + 11.8 \text{ (Average number of syllables per word)}$$

(Losa, 1983:5)

Another measure of the understandability of semantic statements is Gunning's Fog Index of Readability. This index "measures the ease of reading a document based on syntactic

properties of the text" (Farbey, 1990:64). It is important to note that this is not an absolute measure of readability but, rather, "an index ... that shows changes in [the] magnitude" of readability (Farbey, 1990:64). Gunning's Fog Index was originally published in his book The Technique of Clear Writing. The Fog Index, as described by Eisenberg, is calculated using the following procedure:

> (1) Divide the number of words [in the passage] by the number of sentences. This yields the average number of words per sentence.
>
> (2) Count words of three or more syllables (except for proper nouns). Divide this number by the total number of words in the [passage]. The answer is the percentage of difficult words in the [passage].
>
> (3) Add the average number of words in a sentence to the percentage of difficult words.
>
> (4) Multiply the total by 0.4. This gives a Fog count. A count of 10 means the passage should be easy reading for the average tenth grader. (Eisenberg, 1982:290)

A third measure of the understandability of requirement specifications is proposed by Ramamoorthy. Ramamoorthy proposes using several metrics to measure how understandable a specification is "because a single metric cannot cover every aspect of the [software system]." He also states "that the measured values may be objective, but usage of them should be subjective." These measures include the number of functions specified (for example, lines of statement of formal specification language, number of states in state transition model), the connectivity of functions, the amount

of data processed, the connectivity of data, and other various measures. (Ramamoorthy, 1986:81–82)

Measures of Specification Completeness. (IEEE characteristic (2) complete.) The IEEE Standard identifies the following measure of "the completeness of the software specification during the requirements phase" which can also be "used to identify problem areas within the software specification." Furthermore, the IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software reports that this measure has seen moderate use (as opposed to limited or extensive use) in the software community. (IEEE, 1989:25–26, 89–90; IEEE, 1988:32–33)

> The completeness measure consists of the following primitives:
>
> $B_1$ = Number of functions not satisfactorily defined
>
> $B_2$ = Number of functions
>
> $B_3$ = Number of data references not having an origin
>
> $B_4$ = Number of data references
>
> $B_5$ = Number of defined functions not used
>
> $B_6$ = Number of defined functions
>
> $B_7$ = Number of referenced functions not defined
>
> $B_8$ = Number of referenced functions
>
> $B_9$ = Number of decision points not using all conditions, options
>
> $B_{10}$ = Number of decision points

$B_{11}$ = Number of condition options without processing

$B_{12}$ = Number of condition options

$B_{13}$ = Number of calling routines with parameters not agreeing with defined parameters

$B_{14}$ = Number of calling routines

$B_{15}$ = Number of condition options not set

$B_{16}$ = Number of set condition options having no processing

$B_{17}$ = Number of set condition options

$B_{18}$ = Number of data references having no destination

The measure is implemented as follows:

The completeness measure (CM) is the weighted sum of ten derivatives expressed as:

$$CM = \sum_{i=1}^{10} w_i D_i$$

where for each i=1,...,10, each weight $w_i$ has a value between 0 and 1, the sum of the weights is equal to 1, and each $D_i$ is a derivative with a value between 0 and 1.

To calculate the completeness measure, the definitions of the primitives for the particular application must be determined, and the priority associated with the derivatives must also be determined. This prioritization would affect the weights used to calculate the completeness measure.

Each primitive value would then be determined by the number of occurrences related to the definition of the primitive.

Each derivative is determined as follows:

$D_1 = \frac{(B_2 - B_1)}{B_2}$ = Functions satisfactorily defined

$D_2 = \frac{(B_4 - B_3)}{B_4}$ = Data references having an origin

$D_3 = \frac{(B_6 - B_5)}{B_6}$ = Defined functions used

$D_4 = \frac{(B_8 - B_7)}{B_8}$ = Referenced functions defined

$D_5 = \frac{(B_{10} - B_9)}{B_{10}}$ = All condition options at decision points

$D_6 = \frac{(B_{12} - B_{11})}{B_{12}}$ = All condition options with processing at decision points used

$D_7 = \frac{(B_{14} - B_{13})}{B_{14}}$ = Calling routine parameters that agree with the called routines defined parameters

$D_8 = \frac{(B_{12} - B_{15})}{B_{12}}$ = All condition options that are set

$D_9 = \frac{(B_{17} - B_{16})}{B_{17}}$ = Processing follows set condition options

$D_{10} = \frac{(B_4 - B_{18})}{B_4}$ = Data references that have a destination

The value of the completeness measure is scaled between 0 and 1 by the appropriate weights. A score near 1 is considered better than a score near 0. Those values near zero should be traced to the suspect primitive(s) to highlight any need for change in the software specification. As changes are made to the specification, the incremental specification measure values can be plotted to show if improvements are being made and how rapidly. (IEEE, 1989:89-90; IEEE, 1988:32-33)

Another possible completeness metric is actually a measure of the information content of a specification and is

referred to as the "Bang" measure by its author, DeMarco. If we consider specification information content to be analogous with or related to specification completeness, a measure of the information content of the specification might also provide a measure of the completeness of the specification. Likewise, we might consider the measure of information content to provide an indication of how useful the specification will be during operation and maintenance of the system. Since the components and implementation of this measure are lengthy and complex, they are not presented here. And although a brief review of DeMarco's measure is provided in Appendix A, readers are strongly encouraged to examine DeMarco's work for these details. (DeMarco, 1982:80–81)

Agresti provides a set of measures that are somewhat related to DeMarco's information content measure. Using a requirements representation called the Composite Specification Model (CSM) based in part on DeMarco's work, Agresti experimented with 58 explicit measures. The CSM is comprised of three views and corresponding notations (see Figure 2) and acts as a template for specifying requirements. Agresti's measures are organized according to the three views of the CSM. The measures associated with the functional view include a weighted function count and numerical counts of functional primitives, interfaces, internal arcs, internal data items, system input/output data items, and file

input/output data items. The measures associated with the contextual view include numerical counts of entities, events, relationships, attributes, and value sets. The measures associated with the dynamic view include numerical counts of states and transitions. Although Agresti describes these measures as early indicators of system size and complexity, they may also provide indications of the completeness of specifications. (Agresti, 1984)

| Composite Specification Model | |
|---|---|
| Viewpoint | Notation |
| Functional | Data Flow |
| Contextual | Entity-Relationship |
| Dynamic | State-Transition |

Figure 2. Composite Specification Model (Agresti, 1984)

Boehm hints that several simple, explicit counts may be used to quantify specification completeness. Boehm states that a specification must not have any TBDs (use of the phrase "To Be Determined") nor any nonexistent references to be considered complete. With this in mind, one can speculate that counts of the number of TBDs and nonexistent references in a specification will provide a measure of the completeness of the specification. (Boehm, 1984:76–77)

The technique of cause and effect graphing described in the IEEE Standard and discussed under the previous subheading (Unambiguous) may also be helpful in identifying incomplete requirements. (IEEE, 1989:45–47; IEEE, 1988:17–18)

The measure of requirements compliance, also described in the IEEE Standard and discussed under the previous subheading (Unambiguous), may also aid in identifying and quantifying incomplete requirements. (IEEE, 1[illegible]):70–72; IEEE, 1988:25–26)

Measures of Requirement Verifiability (Testability). (IEEE characteristic (3) verifiable.) According to the IEEE Guide to Software Requirements Specifications, "a requirement is verifiable if and only if there exists some finite cost-effective process [to] check that the software product meets the requirement" (IEEE, 1984:12). In other words, if we can cost-effectively test a requirement, it is verifiable.

Ramamoorthy and others have proposed requirements complexity metrics that can be used to infer the cost-effectiveness, or difficulty, of testing requirements. Ramamoorthy focused on measuring complexity during the requirements phase and developed what is termed "a spectrum of metrics" for requirements written in the control flow requirement specification language RSL. (RSL is based on the control flow and entity-relationship models of software

specification.) The control flow model is implemented through a control flow graph consisting of nodes (specifying processing operations) and connecting arcs. The proposed complexity metrics are based on measures of the essential elements of the specification language and model such as the number of nodes, connecting arcs, and paths in the network. The proposed metrics include several which are specifically identified to infer a measure of the difficulty of testing specifications and, therefore, the verifiability of the specification. Again, since the details of these measures are lengthy and complex, they are not presented here. And, once again, although a brief review of these measures is provided in Appendix A, readers are encouraged to examine Ramamoorthy's work for further details. (Ramamoorthy, 1985:113-115)

Measures of Specification Consistency. (IEEE characteristic (4) consistent.) The IEEE Standard Dictionary of Measures to Produce Reliable Software describes a simple measure of the number of conflicting requirements.

> This measure is used to determine the reliability of a software system, resulting from the software architecture under consideration, as represented by a specification based on the entity-relationship-attribute model. (IEEE, 1989:53; IEEE, 1988:21)

It is implemented and interpreted as follows:

> The mappings from the software architecture to the requirements are identified. Mappings from the same specification item to more than one differing

> requirement are examined for requirements inconsistency. (If the same specification item maps to two different requirements items, the requirements should be identical. Otherwise, the requirements are inconsistent.) Mappings from more than one specification item to a single requirement are examined for specification inconsistency. (IEEE, 1989:53-54; IEEE, 1988:21)

The measure of requirements compliance described in the IEEE Standard and discussed under the previous two subheadings (Unambiguous and Complete) may also aid in identifying and quantifying inconsistent requirements. (IEEE 1989:70-72; IEEE, 1988:25-26)

Measures of Specification Modifiability. (IEEE characteristic (5) modifiable.) One criteria for modifiability defined in the IEEE Guide to Software Requirements Specifications is that "the same requirement should not appear in more than one place in the [specification]" since redundancy can easily lead to inconsistencies (IEEE, 1984:12). Therefore, any of the measures and techniques which help identify redundant or inconsistent requirements (for example, some of the measures discussed under the previous subheading (Consistent)) may also aid in measuring the modifiability of specifications.

Ramamoorthy has proposed several dependency metrics for requirements written in the control flow requirement specification language RSL that can be used to measure "the dependency of parts of the software on other parts"

(Ramamoorthy, 1985:114). These dependency metrics may provide indications of the modifiability of the specification.

> The dependency metrics measure the dependency of parts of the software on other parts. The greater this dependency the more the chance that modification of a part of the software due to a bug will lead to other bugs in dependent parts of the program. This is the ripple effect. One metric for this is the number of requirement networks (R_NETs) that are enabled directly or indirectly through a sequence of other R_NETs. (Ramamoorthy, 1985:114)

Measures of Requirement Traceability. (IEEE characteristic (6) traceable.) The IEEE Standard describes an extensively used measure which "aids in identifying requirements that are either missing from, or in addition to, the original requirements." This measure is implemented and interpreted as follows:

> A set of mappings from the original requirements is created. Count each requirement met by the architecture (R1) and count each of the original requirements (R2). Compute the traceability measure (TM):
>
> $$TM = \frac{R1}{R2} \times 100\%$$
>
> When all of the original software requirements are covered in the software architecture, the traceability measure is 100%. A traceability measure of less than 100% indicates that some requirements are not included in the software architecture. (IEEE, 1989:47-48; IEEE, 1988:18)

Measures of Specification Usability During Operation and Maintenance. (IEEE characteristic (7) useable during the

operation and maintenance phase.) The IEEE Standard identifies a measure of the quality of software documentation and source listings. This measure, determined through the use of questionnaires, may identify the areas of any software product which might be inadequate for use in a software maintenance environment. It is described in the IEEE Standard as follows:

> Two questionnaires, the Software Documentation Questionnaire and the Software Source Listing Questionnaire, are used to evaluate the [format and content of] software products in a desk audit [from a maintainability perspective]. The questionnaires are contained in Software Maintainability Evaluation Guide. The guide can be ordered from the Air Force Operational Test and Evaluation Center. (IEEE, 1989:83-84; IEEE, 1988:29)

The measure of information content of a specification, referred to as the "Bang" measure and discussed under an earlier subheading (Complete), may also provide an indication of how useful the specification will be during operation and maintenance of the system. For example, a specification with a high measure of information content might be more useful when performing maintenance than a specification with a low measure of information content. (DeMarco, 1982:80-81)

Checklist and Worksheet Measures of Specification Quality. One popular technique of measuring the quality of a software requirement specification is not appropriately listed under any of the characteristics of a good specification listed above, since it really measures the

quality of the document as a whole. This technique involves using checklists or worksheets in a complete review or inspection of the specification. "A checklist is a list of the properties of the software that together determine whether or how far the criteria have been met" (Farbey, 1990:64). More specifically, checklists are "specialized lists, based on experience, of significant issues [required to insure] successful software development" that are compared against a specification in order to verify the specification adequately addresses those issues (Boehm, 1984:80). Worksheets are primarily used to translate specific measurements of items on a checklist into a metric score. The distinction is that worksheets are used to compute a metric score whereas checklists are not. Several very comprehensive sets of checklists and worksheets have been published by the Air Force Systems Command's Electronic Systems Division and Rome Air Development Center. Two of these are the four volume Computer Systems Acquisition Metrics Handbook prepared by Systems Architects, Inc. and the three volume Software Quality Measurement for Distributed Systems technical report prepared by Boeing Aerospace Company (Boeing Aerospace Company, 1983; Systems Architects, Inc., 1982).

## Conclusion

It is important to note that the metrics presented here are comprised of software metrics specifically developed to measure features of the requirements analysis process and products, as well as other measures and measurement techniques that may be of value during requirements analysis.

Whether a measure is developed for a specific purpose or if it is being tested in a new application, the successful use of metrics depends on the user's enforcement of a disciplined data collection process and the serious review of the data collected for each metric. When properly implemented, metrics can provide early indications of potential software development problems and can call attention to and stimulate discussion leading to early resolution of those problems. (Schultz, 1988:1)

The metrics identified in this literature review and the references in which they may be found are summarized on the next page in Table 1.

**TABLE 1**

**Candidate Requirement Metrics**

| Title | Reference |
|---|---|
| Fault-Days Number | IEEE, 1989:42–43<br>IEEE, 1988:16–17 |
| Error Distribution Measure | IEEE, 1989:49–51<br>IEEE, 1988:19 |
| Manhours Per Major Defect Detected | IEEE, 1989:52–53<br>IEEE, 1988:19 |
| Defect Index | IEEE, 1989:48–49<br>IEEE, 1988:18–19 |
| Schedule Progress | Schultz, 1988:22 |
| Ambiguity Poll | Gause, 1989:217–224 |
| Misinterpretation and Comprehension | Cioch, 1991:86–87 |
| Cause and Effect Graphing | IEEE, 1989:45–47<br>IEEE, 1988:17–18 |
| Requirements Compliance | IEEE, 1989:70–72<br>IEEE, 1988:25–26 |
| Flesch-Kincaid Readability Formula | Losa, 1983:5 |
| Gunning's Fog Index of Readability | Eisenberg, 1982:290 |
| Specification Completeness | IEEE, 1989:89–90<br>IEEE, 1988:32–33 |
| "Bang" – A Functionality Measure | DeMarco, 1982:80–81 |
| Composite Specification Measures | Agresti, 1984 |
| Control Flow Measures | Ramamoorthy, 1986:81–82<br>Ramamoorthy, 1985:113–115 |
| Number of Conflicting Requirements | IEEE, 1989:53–54<br>IEEE, 1988:21 |
| Requirements Traceability | IEEE, 1989:47–48<br>IEEE, 1988:18 |
| Software Documentation and Listings | IEEE, 1989:83–84<br>IEEE, 1988:29 |
| Explicit Count(s) Metrics | Boehm, 1984:76–77<br>Fouser, 1988:6<br>Ramamoorthy, 1986:81–82<br>Ramamoorthy, 1985:113–115 |
| Checklist and Worksheet Measures | Systems Architects, 1982<br>Boeing Aerospace Co., 1983 |

# III. Methodology

This chapter describes the methodology used to obtain and analyze the data collected during this study. It includes a discussion of the research methodology followed, the survey instrument used, the data collection process, and the data analysis techniques used in performing this study.

## Research Methodology

As stated in Chapter II, the research methodology is based in part on Basili and Rombach's goal/question/metric (GQM) paradigm. The GQM paradigm consists of three steps (Basili, 1987:350-351; Shepperd, 1990:312). All three are completed in this study. Incidentally, in performing these steps, the first three investigative questions put forth in Chapter I are answered.

The first step is to identify a goal or set of goals. In most cases the goal is to improve a process or product. For this study, the overall goal was to improve the requirements analysis phase of software development. Although this goal is very admirable, it is vague and is better represented by two less ambitious, more specific goals that must be met in order to achieve the overall goal. These

specific goals, identified in response to investigative question number one, were determined to be:

(1) To assess the productivity and quality of the requirements analysis process.

(2) To assess the quality of the products being produced by that process. (Dziegiel, 1991)

The second step of the GQM paradigm involves formalizing the goals into questions that, if answered, result in realization of the goals (Basili, 1987:350–351; Shepperd, 1990:312). In accomplishing this step, these questions were found to be:

(1) Does the method in which requirements are solicited from users, specified, and validated against the original intent of the user have a significant effect on software quality?

(2) Which requirements analysis and specification practices produce the highest quality products? Which produce the lowest quality products?

(3) What percentage of errors found throughout development, testing, and operation are due to poor requirements analysis and specification? (Dziegiel, 1991)

These questions are offered as a response to investigative question number two.

In the third and last step, metrics are developed or identified to provide the information needed to answer the questions posed in step two (Basili, 1987:350–351; Shepperd, 1990:312). Since the author does not have the expertise to develop metrics specifically for this purpose, an alternative solution had to be found. Consequently, an extensive literature review was performed to identify metrics that might provide this information. The collection of metrics resulting from this review are presented in Chapter II. This collection was assembled in response to investigative question number three.

Since the metrics identified in the literature review collectively held only a small likelihood of providing the information needed to answer our questions and, subsequently, fulfill our goals, further research was required. In order to more positively determine which metrics should provide this information, an analysis of the perceptions that software professionals have of the utility of various metrics identified in the literature review was performed. This analysis focused on the perceptions that software professionals have of the utility of various metrics in assessing the productivity of the requirements analysis

process and the quality of the products produced by this process.

## Justification of Approach

A survey and semi-structured interview were used to collect data of software professionals' perceptions of the utility of various requirement metrics. These perceptions provide indications of the usefulness of these metrics when applied to actual requirements analyses. A survey was used to gather initial data and a follow-up interview was selectively used to further investigate significant aspects of the survey data.

This approach was chosen because no other data was readily available or could be obtained to adequately fulfill the research objectives. A better approach would have been to collect data from actual experiences with the various requirement metrics. However, since these metrics are only in very limited use (if they are being utilized at all), the author was not able to locate data of this kind.

## Survey Instrument

The survey questionnaire consisted primarily of a series of six questions designed to collect information about the

utility of selected software metrics when applied to software requirements analyses. This series of questions was asked for each metric included on the questionnaire. (The questionnaire is provided as Appendix A.) Additionally, three questions were asked to determine the level of experience each participant had in performing requirements analyses and their overall experience with software development. This information was used as the basis for an analysis of the validity of the data collected.

The questionnaire was designed to determine the metrics that would be useful during the requirements analysis phase of software development. The questionnaire included both process and product metrics that could be applied to, respectively, the requirements analysis process and formal or informal software requirements specifications.

There were 18 metrics selected for use in the questionnaire including 5 process metrics, 10 product metrics, 1 collection of miscellaneous explicit product and process measures (grouped together for the sake of brevity), and 2 worksheet/checklist type measures. Of the 10 product measures, 4 utilized graphical techniques and 4 were applicable only to formal requirement specifications.

The selections were based on several criteria. The first criterion was a desire to include a suitable number of

process and product metrics without the total number becoming burdensome upon the participants. The second criterion was a desire to include the broadest possible spectrum of different types of measures; for example, metrics intended to measure different product qualities, metrics applicable only to formal specifications, and metrics applicable only to informal specifications.

The third criterion was the necessity to include an appropriate number of metrics that have been approved by the Institute of Electrical and Electronics Engineers Standards Board; i.e., metrics included in the IEEE Standard Dictionary of Measures to Produce Reliable Software. These were included so that responses regarding these metrics could be compared with the responses regarding the other metrics and with IEEE data about experiences with these metrics. This information provided an essential standard with which to judge the quality of the survey data.

The last criterion was a desire to include a metric that had little scientific basis in order to provide another benchmark to compare the data with. This untested measure would provide a benchmark just as the IEEE measures would, except at the other end of the experience scale. For this reason the author created a simple process measure based on a ratio of the number of requirements already specified to the estimated total number of requirements to be specified. This

measure was also similar to the indicator of requirements stability suggested in Air Force Systems Command Pamphlet 800-43 (Department of the Air Force, 1990:11). The measure took the form:

$$\text{Progress} = \frac{\text{Number of requirements documented}}{\text{Estimated total number of requirements}} \times 100\%$$

This measure was named the Requirements Documentation Progress metric and was included on the questionnaire.

A list of the metrics included on the questionnaire and the references they were taken from is provided on the following page in Table 2.

The series of six questions included on the questionnaire were not actually questions, but rather statements that the respondents could agree or disagree with. In order to collect data to determine the relative ranking of each metric's utility, the participants were asked to select an appropriate response to each statement according to a five point Likert scale (Emory, 1980:271–274). An example of the scale is provided in the following figure.

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

Figure 3. Example of Scale Used on Questionnaire

**TABLE 2**

**Requirement Metrics Included on the Questionnaire**

| Title | Reference |
|---|---|
| Schedule Progress | Schultz, 1988:22 |
| Requirements Documentation Progress | Created for this study |
| Fault-Days Number | IEEE, 1988:16–17 |
| Error Distribution Measure | IEEE, 1988:19 |
| Manhours Per Major Defect Detected | IEEE, 1988:19 |
| Requirements Understandability | Losa, 1983:5 |
| Requirement Ambiguity | Gaus[illegible], 1989:217–224 |
| Requirements Traceability | IEEE, 1988:18 |
| Number of Conflicting Requirements | IEEE, 1988:21 |
| Requirements Compliance | IEEE, 1988:25–26 |
| Specification Completeness | IEEE, 1988:32–33 |
| Cause and Effect Graphing | IEEE, 1988:17–18 |
| "Bang" – A Functionality Measure | DeMarco, 1982:80–81 |
| Composite Specification Measures | Agresti, 1984 |
| Control Flow Measures | Ramamoorthy, 1986:81–82<br>Ramamoorthy, 1985:113–115 |
| Miscellaneous Explicit Counts | Boehm, 1984:76–77<br>Fouser, 1988:6<br>Ramamoorthy, 1986:81–82<br>Ramamoorthy, 1985:113–115 |
| Completeness Checklist | Systems Architects, 1982 |
| Requirements Analysis Worksheet | Boeing Aerospace Co., 1983 |

Five of the six statements parallel the features of an ideal software metric identified in the Software Engineering Institute (SEI) curriculum module on software metrics. According to the SEI, "ideal metrics should be simple,

precisely definable, objective, easily obtainable, valid, and robust" (Mills, 1988:4). These statements were designed to allow the participant to rate each metric according to specific and independent factors. It was hoped that this approach would limit bias in the data. The sixth statement was intended to determine the participant's overall assessment of the utility of the metric and could not be designed to limit bias in the responses. The six statements are:

(1) This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.)

(2) The data needed to calculate this metric is easily obtained before or during requirements analysis.

(3) The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it.

(4) This metric measures the quality intended to be measured.

(5) This metric is insensitive to small changes in the requirements analysis process or product (as applicable).

(6) This metric would be useful during requirements analysis.

Additional comments about each metric were invited from the participants. Space was provided on the questionnaire for this purpose.

## Data Collection

Data was collected using a three step process. The first step involved identifying possible survey participants. Originally, the survey was intended to include practicing software engineers and computer scientists, persons serving in a software engineer's or computer scientist's position, software systems project managers, persons teaching courses in a software engineering or computer science curriculum, and persons performing research in the area of software metrics. However, in order to keep the validity of the data collected as high as possible, only software professionals with experience performing requirements analysis and a fair knowledge of software metrics were surveyed. Due to these rather stringent requirements, only a limited number of persons from the original sample were identified, and would be asked to participate. Furthermore, since only a limited number of persons were surveyed (27 questionnaires were sent to consenting participants and less than half responded),

this research should be considered a "pilot study" for further research in this area.

Next, the participants identified in the first step were contacted in person or by telephone, to discuss some of the details of the study and to gain their consent to take part in the study. When this permission was obtained, a questionnaire was mailed to the participant.

Finally, follow-up interviews were conducted with several participants to further investigate significant aspects of the data collected through the survey. In these cases, a semi-structured interview was used to probe for the additional information needed to better understand the original data. An example of the list of items discussed during an interview is provided in Appendix B.

## Data Analysis

Since only a limited number of persons were surveyed and, subsequently, a limited amount of data collected, detailed analysis of the data using statistical tests was not appropriate. However, an analysis using descriptive statistics was performed. This analysis was intended to answer the remaining four investigative questions put forth in Chapter I.

Descriptive statistics are commonly used to characterize data. In this study, several descriptive statistics were used to present and analyze the data including the response mean and standard deviation. The mean response to a particular statement was used to compute scores in order to rank-order the metrics according to the participants' perceptions of the utility of each metric. The response standard deviation was used to measure the participants' level of agreement with statements concerning their perceptions of the usefulness of each metric.

Mean scores of the responses to each statement and a ranking of each metric's utility were determined using two mathematical equations. Equation (1) was used to compute a mean score for the responses to each of the six statements.

$$\text{MEAN SCORE} = \frac{(5A + 4B + 3C + 2D + E)}{N} \qquad (1)$$

where: A = number of Strongly Agree responses
B = number of Agree responses
C = number of Neither Agree Nor Disagree responses
D = number of Disagree responses
E = number of Strongly Disagree responses
N = total number of responses

Equation (2) was used to compute an overall score for each metric. The overall score was used to determine the relative

ranking of the metrics on the questionnaire based on the participants' perceptions of the usefulness of each metric.

$$\text{OVERALL SCORE} = MS_1 + MS_2 + MS_3 + MS_4 + MS_5 + 2MS_6 \quad (2)$$

where: $MS_1$ = mean score on statement #1
$MS_2$ = mean score on statement #2
$MS_3$ = mean score on statement #3
$MS_4$ = mean score on statement #4
$MS_5$ = mean score on statement #5
$MS_6$ = mean score on statement #6

Since the objective of this study was to determine the perceived usefulness of various metrics, the author felt that a participant's overall assessment of the utility of each metric should be of greater importance in the ranking than his assessment of the specific qualities of the metric. For this reason the contribution of the score for statement six is twice the contribution of the scores for each of the other statements.

Additionally, the data was reviewed to determine if a significant correlation existed between the software professionals' experience and their perception of any or all of the requirement metrics' utility. Once again, although information about the experience level of the participants was collected in the survey, no attempt was made to determine the competency of the participants.

Finally, the data was inspected to determine if some common thread could be found among the responses that reflected any significant relationships that may have been overlooked.

## Conclusion

This chapter described the methodology used to obtain and analyze the data collected during this study. An analysis of the data collected during this research effort and a discussion of the findings of this study is provided in the next chapter.

# IV. Findings

This chapter presents the results obtained using the methodology described in the previous chapter. The demographics of the respondents are presented first and then each of the seven investigative questions are answered.

## Demographic Information

Just under half of the individuals who agreed to participate responded; only 13 of 27 questionnaires were returned. The experience levels and professional responsibilities of the respondents are shown in Table 3.

**TABLE 3**
**Respondent Demographic Information**

| | | |
|---|---|---|
| Total number of respondents: **13** | | |
| Years involved with software development: | | |
| 5 or less: **0** | Between 5 & 10: **2** | 10 or more: **11** |
| Number of times involved with software requirements analyses: | | |
| 5 or less: **3** | Between 5 & 10: **3** | 10 or more: **7** |
| Current professional responsibilities: | | |
| Educational: **8** | SW Development: **2** | SW Research: **3** |

One point of particular interest is the high ratio of respondents with educational responsibilities to respondents with software development or research responsibilities. Approximately equal numbers of each category agreed to participate and were sent questionnaires. However, the response rate for individuals with educational responsibilities was much higher than the response rates of the other categories.

Another important point is that the experience levels of all of the participants are relatively high. This is primarily a result of the screening process used to select the participants. As mentioned in Chapter III, in order to insure the validity of the data collected was as high as possible, only software professionals with experience performing requirements analysis and a fair knowledge of software metrics were surveyed. The success of the screening process is apparent in the demographics. However, the effort to keep the data valid also caused a problem in answering the last investigative question. This unexpected problem is discussed further in the answer to investigative question seven.

The last point of discussion is that the total number of participants is relatively small, as was expected. The small number of respondents does not provide enough data to draw

statistically significant conclusions about the utility of any or all of the metrics. However, a qualitative analysis of the data can be and was performed. This analysis is presented in the following sections.

## Investigative Questions 1 - 3

The first three investigative questions were based on the goal/question/metric paradigm proposed by Basili and Rombach (Basili, 1987:350-351; Shepperd, 1990:312):

(1) What are the specific goals for improving the requirements analysis phase of the software development life cycle?

(2) What questions can quantify these goals?

(3) What metrics might provide the information needed to answer these questions?

The answers to these questions were presented in the Research Methodology section of Chapter III and are not repeated here.

## Investigative Question 4

The purpose of investigative question four was to determine the relative ranking, in terms of perceived utility, of the software metrics available for use during

requirements analysis. The rankings and the overall scores used to compute the rankings, shown on the next page in Table 4, were determined using the equations discussed previously in the Data Analysis section of Chapter III. However, as shown, the differences between the overall scores are trivial and, therefore, the rankings are meaningless. In other words, the respondents, as a group, believe the utility of all of the metrics are about equal. No metric is perceived as more or less useful than any other metric. This perception is also reflected in the responses to survey statement number six. (Statement six allowed the respondents to directly state their opinions of the utility of each metric.) For the sake of comparison, a ranking of the metrics according to the mean scores for statement six are also shown in Table 4. (The mean scores for all survey statements are provided in Appendix C.)

Some of the similarities in the two sets of rankings can be attributed to the scoring system used to rank-order the metrics. The overall scores were calculated using "two parts" of the mean score for survey statement six and "one part" each of the mean scores for statements one through five. Thus, the rankings by overall score are influenced twice as much by the responses to survey statement six as by the responses to the other statements.

**TABLE 4**

**Relative Ranking of Metrics**

**Included on the Questionnaire**

| Title of Metric | Ranking by Overall Score (Overall Score) | Ranking by Statement 6 Mean Score (Mean Score) |
|---|---|---|
| Error Distribution Measure | 1 (23.3) | 9 * (3.2) |
| Miscellaneous Explicit Counts | 2 (23.2) | 2 * (3.5) |
| Manhours Per Major Defect Detected | 3 (23.0) | 2 * (3.5) |
| Requirements Analysis Worksheet | 4 (22.5) | 1 (3.7) |
| Requirement Documentation Progress | 5 * (22.3) | 6 * (3.3) |
| Completeness Checklist | 5 * (22.3) | 2 * (3.5) |
| Control Flow Measures | 7 (21.8) | 2 * (3.5) |
| Schedule Progress | 8 (21.3) | 12 * (2.8) |
| Requirements Traceability | 9 * (21.0) | 9 * (3.2) |
| Requirements Compliance | 9 * (21.0) | 6 * (3.3) |
| Specification Completeness | 11 (20.8) | 9 * (3.2) |
| Cause and Effect Graphing | 12 * (20.2) | 6 * (3.3) |
| Fault-Days Number | 12 * (20.2) | 12 * (2.8) |
| "Bang" - A Functionality Measure | 14 (18.8) | 12 * (2.8) |
| Composite Specification Measures | 15 (18.3) | 15 * (2.7) |
| Requirements Understandability | 16 (18.0) | 18 (1.8) |
| Number of Conflicting Requirements | 17 (17.2) | 16 (2.5) |
| Requirement Ambiguity | 18 (15.0) | 17 (2.3) |

**NOTES:** 1. * signifies a tie in ranking.
2. Overall and mean scores used to determine the rankings were calculated only to within a $10^{th}$ of a point; i.e., 2.5 points.

The finding that the rankings are inconclusive is supported in that the overall scores are generally within one standard deviation of each other. Furthermore, the overall scores and mean scores for survey statement six are generally all in a range of values corresponding to an indifferent (neither agree nor disagree) response. This range is approximately equal to an overall score of 17–25 and a mean score for statement six of 2.5–3.5. A plot of the overall scores is provided in Figure 4 and a plot of the mean scores for statement six is provided in Figure 5. The plots include error bars equal to the standard deviation for each score in order to display the trivial differences between overall scores and between mean scores for statement six. Where the error bars overlap for scores of two or more metrics, one score cannot be considered significantly larger or smaller than another score. These plots make it readily apparent that the relative rankings are questionable.

One additional finding is apparent in Figure 6. In Figure 6, the average mean scores for survey statements one through five and the mean scores for statement six are plotted together for comparison. It is fairly evident that a correlation exists between the two sets of mean scores. (Recall that statements one through five were used to determine the participants' opinions of how each metric compared to the five qualities of an ideal metric and

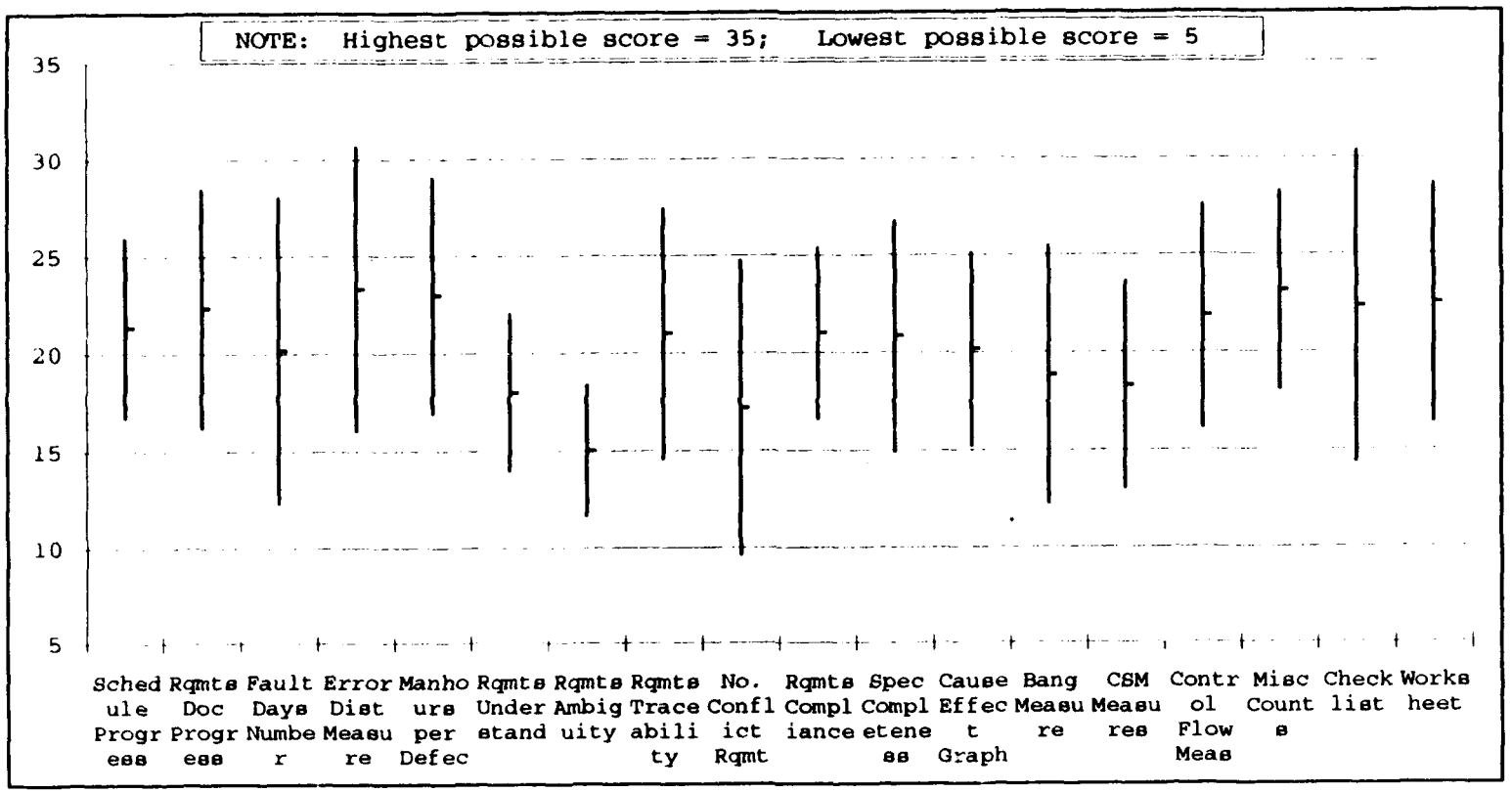



Figure 4. Overall Scores (Perceived Metric Utility)

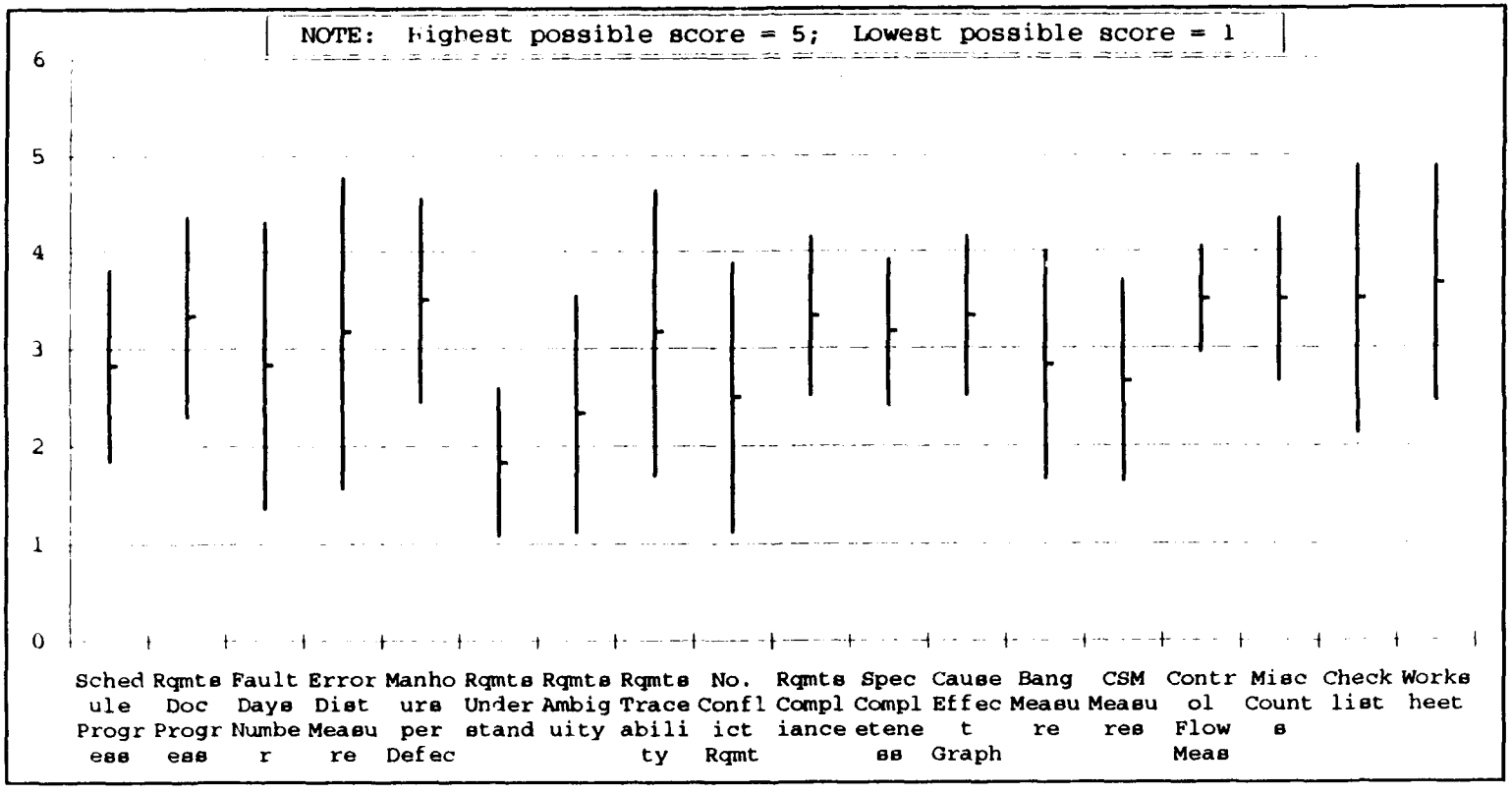



Figure 5. Questionnaire Statement #6 Mean Scores (Overt Opinion of Metric Utility)

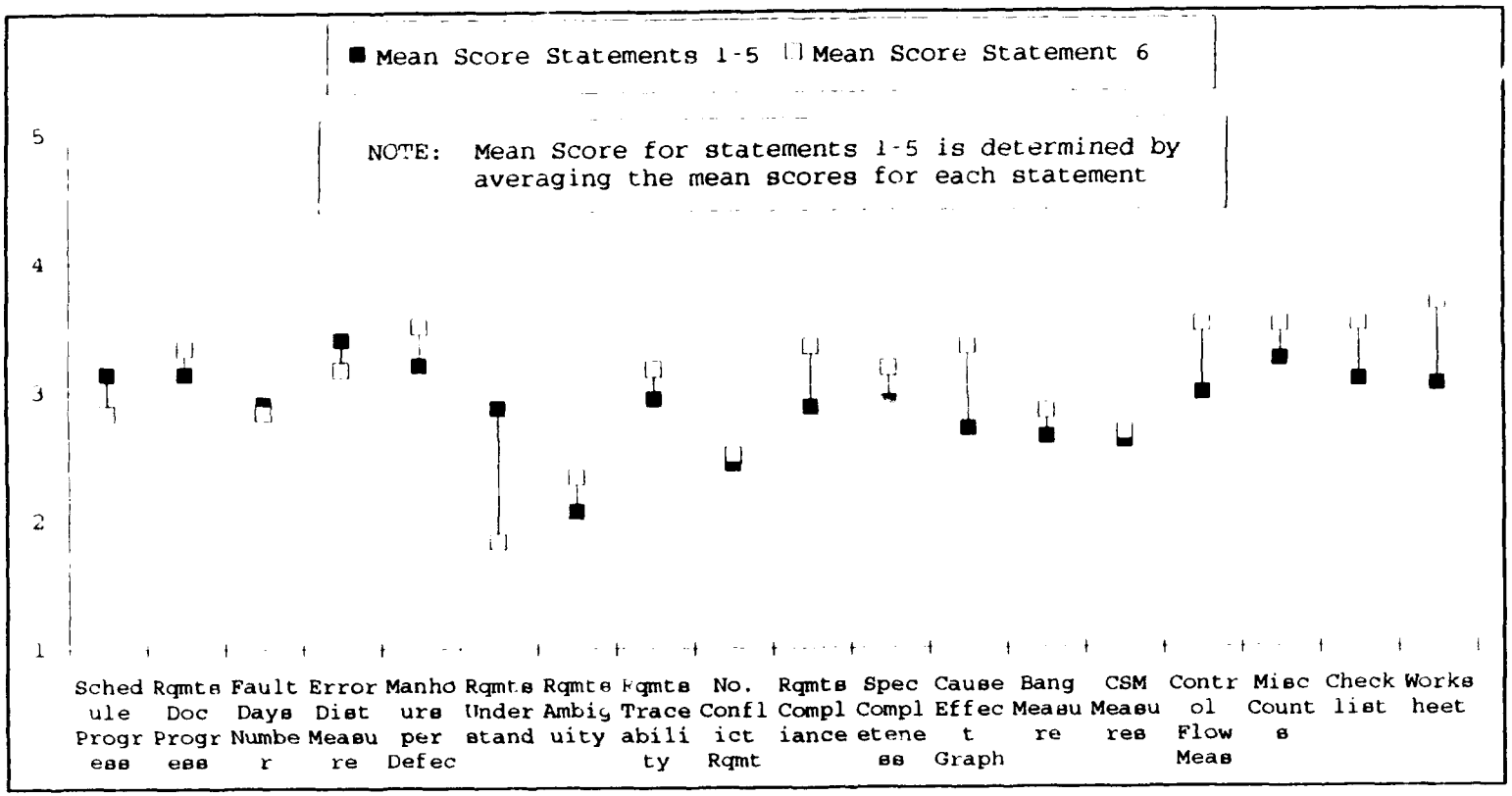



Figure 6. Questionnaire Statements #1–5 Mean Scores vs Statement #6 Mean Scores

statement six was used to allow the participants to directly state their opinions of the utility of each metric.) It is gratifying to see that the participants' overt opinions of the usefulness of the metrics appear to match their opinions of how the metrics compare to the qualities of good metrics. Although the correlation shown in Figure 6 is not statistically significant and does not prove anything, it does indicate that the six statement methodology was probably sound.

**Investigative Question 5**

The purpose of this investigative question was to determine which requirement metrics, if any, were perceived as significantly more or significantly less useful than other metrics. As presented in the previous section, the data does not indicate that any of the metrics were perceived to be significantly more useful than the others. Once again, the respondents generally were indifferent as to the utility of the metrics on the questionnaire.

However, there is an indication that the measures of requirements understandability, requirement ambiguity, and the number of conflicting requirements are perceived to have the least utility of all the metrics on the questionnaire. These measures placed in the bottom of the rankings and had

overall scores corresponding to the bottom of the indifferent range. In other words, the respondents, as a group, had slightly negative perceptions of the utility of these measures. The rankings and scores for these metrics are provided below in Table 5.

**TABLE 5**

**Least Useful Requirement Metrics**

| Title of Metric | Overall Ranking | Overall Score | Statement 6 Mean Score |
|---|---|---|---|
| Requirement Ambiguity | 18 | 15.0 | 2.3 |
| Number of Conflicting Requirements | 17 | 17.2 | 2.5 |
| Requirements Understandability | 16 | 18.0 | 1.8 |

During the follow-up interviews, several participants provided the principal reasons lower scores were assigned to these measures:

Requirements understandability: "The formula cannot possibly be reliable for highly technical specifications."

Requirement ambiguity: "This metric is too dependent on the wording of the questions used in conducting the poll and, therefore, would be too subjective to be useful."

Number of conflicting requirements: "This metric is too difficult to perform."

**Investigative Question 6**

This question was intended to identify any significant differences between the rankings for product and process type metrics. No significant differences were found. A review of Table 4, Figure 4, and Figure 5 indicates that the five process metrics included on the questionnaire (schedule progress, requirements documentation progress, fault-days number, error distribution measure, and manhours per major defect detected) are perceived to have the about the same utility as the product metrics.

**Investigative Question 7**

The last investigative question was intended to identify any significant correlations between a software professional's experience and the perception of the utility of a particular requirement metric. Due to the screening process mentioned earlier, only relatively highly experienced personnel were surveyed. A lack of relatively unexperienced participants make it impossible to determine if any statistically significant correlation exists.

However, comments written on the questionnaire and made during follow-up interviews provide some rather meaningful

information. These comments, provided by the respondents with the most experience with software metrics and summarized here, referred to the formality and preciseness of the metrics on the questionnaire. These respondents point out that metrics that are precisely defined and have detailed descriptions of the use of the metrics would be considered more useful than metrics that are not precisely or formally defined. This sentiment reflects the respondents' beliefs that they would be more confident about the utility of metrics that are precisely defined as opposed to metrics with only an implied use. These beliefs are revealed in one respondent's comments: "The metrics presented are not defined formally or precisely. ... The need for formalism [must be stressed]."

## Conclusion

This chapter presented the basic findings resulting from performing this study. A discussion of the significant conclusions that can be drawn from these findings is provided in the next chapter.

# V. Conclusions and Recommendations

This chapter provides a summary of this study, presents the significant conclusions that are derived from the research findings, and offers several recommendations for revised and follow-on research.

## Project Overview

This study was performed in an attempt to gather information about the usefulness of various software requirement metrics. A broader goal was to provide some of the information needed to make intelligent choices of requirement metrics on which to focus further research efforts and, more importantly, to use on future software developments.

The study employed a two part methodology. In the first part, the specific goals of the requirements measurement effort and the requirement metrics worthy of further investigation were identified. In the second part, a survey was conducted to determine the perceptions that software professionals have of the utility of several metrics selected from those identified in part one.

## Conclusions

The findings presented in the previous chapter support only one significant conclusion. Unfortunately, this conclusion does not directly correspond to the research objective put forth earlier in this study.

The respondents, as a group, emphatically expressed the opinion that a metric must be precisely defined for it to be considered useful. The respondents claimed that, unless a metric is precisely and formally defined and has a detailed description of how to use it (implement it), they would be skeptical about its utility.

This claim is not unreasonable. Kitchenham, Pickard, and Linkman have stated that "it is clear that we must improve metrics definitions if metrics are to be properly validated" and later used on software developments (Kitchenham, 1990:57). It is unmistakable that the respondents agree with this opinion. It is also evident the data collected in this study confirms the validity of Mill's first criteria for an ideal metric; the requirement for metrics to be "simple, precisely definable—so that it is clear how the metric can be evaluated" (Mills, 1988:4).

## Closing Discussion

Several aspects of the findings deserve further discussion and are reviewed here. First, it should be made clear that the rankings derived from the data and presented in this study are inconclusive. The data does not indicate that any significant differences between the utility of the selected requirement metrics exist. For example, eight of the metrics included on the questionnaire are recommended by the IEEE for use on software developments but were ranked no higher or lower than the other measures. (One would expect the IEEE recommended measures to be ranked at least slightly higher than the other measures.) In general, the respondents were indifferent as to the utility of the metrics on the questionnaire and reiterated the mixed opinions that software professionals have of the usefulness of software metrics (Mills, 1988:17). Furthermore, one metric, created by the author as a means to help validate the data, ranked higher than six of the IEEE approved metrics, including one with "extensive experience in industry" (IEEE, 1989:26). Although the IEEE experience ratings are caveated with the statement "in no way does the experience rating imply that one measure is better than another," the author wonders how a new measure can actually be more useful than several other proven

measures (IEEE, 1989:25). For these reasons the rankings should not be considered, in any way, conclusive.

Secondly, much of this study was based on an article by Farbey in which he identified various metrics for use during requirements analysis. Many of Farbey's recommended measures were investigated in the course of this study. In concluding his article, Farbey points out that "it may be that metrics are not appropriate" for use with informal specifications (Farbey, 1990:64). Since metrics used to measure the qualities of informal specifications are difficult to define and implement, and since this study found that metrics must be precisely and formally defined to be useful, this study may help prove him to be correct.

On the other hand, measures of the characteristics of formal specifications such as Ramamoorthy's control flow measures may, in fact, be very useful. Although the respondents were generally indifferent about the utility of all of the requirement metrics, the respondents' indicated (in their responses to survey statement six) that they believed the control flow measures could be more useful than other measures. This could be interpreted as an indication that the respondents believe measures of the characteristics of formal specifications are more useful than other metrics. If this interpretation is correct, it would support similar

views expressed by Agresti and Ramamoorthy. (Agresti, 1984; Ramamoorthy, 1986:75–83; Ramamoorthy 1985:111–120)

Additionally, the National Aeronautics and Space Administration's Jet Propulsion Laboratory (JPL) has been collecting data of explicit measures of software since the 1970s including several explicit measures of the characteristics of software requirements specifications (Bush, 1989:iii; Fouser, 1988:6). After compiling and analyzing this data "JPL now has a rough measurement foundation for software productivity and software quality and an order-of-magnitude quantitative baseline for software systems" (Bush, 1989:26). It appears that JPL has made progress in using quantitative data derived from explicit metrics, including requirement metrics, to measure software productivity and quality (Bush, 1989:28). On the other hand, Agresti experimented with 29 objective measures such as the explicit counts of number of pages, number of input/output requirements, number of constraints, and found that "the simple counts ... were not useful measures because they reflect the variability that is found in the representation of requirements" (Agresti, 1984). These efforts provide two benchmarks with which other experiences with simple, explicit measures may be compared.

Another benchmark may be provided by Ramamoorthy's experiments with his control flow measures on a development

at the University of California at Berkeley (Ramamoorthy, 1985:120). Even though details regarding Ramamoorthy's experiments were not found during the course of this study, data may be available in the future for use on new efforts with requirement metrics.

Furthermore, it is also evident the data collected in this study confirms a fairly common opinion that readability formulas such as the Gunning Fog Index or the Flesch-Kincaid formula "are not particularly useful to the technical writer" (Riney, 1989:56–57, 208–209). Riney asserts that "the readability [of a technical document] is important; however, the degree of accuracy of the [technical document] can mean the difference between" a reader successfully interpreting the text or misinterpreting it. It may be that precise technical writing, such as the kind required in software requirements specifications, is incompatible with truly readable writing.

Although the conclusions of this study emphasize precise and, therefore, objective metrics, subjective measures may still be useful and should not be disregarded. For example, Ross suggests that "subjective metrics are unreliable for preliminary identification of anomalies but very useful when diagnosing their causes" (Ross, 1990:85).

Finally, notwithstanding the inconclusive results of this study, requirement metrics can be very useful. Sheppard asserts:

> The use of metrics derived from specifications and designs is starting to be developed as a means of obtaining early feedback [on software quality]. ... It may be argued that this has profound consequences on the way in which software development decisions can be made. Certainly, it augments the traditional decision making by guesswork or by analogy. (Sheppard, 1990:311)

## Recommendations

It is recommended that this research be revised and repeated. A revised study involving a similar methodology should include the following changes:

(1) Define the metrics on the questionnaire more precisely and/or replace them with more precisely defined metrics.

(2) Provide detailed descriptions of how to use (i.e., implement) the metrics.

(3) Reduce the number of metrics included on the questionnaire in order to reduce the burden on the participants and to insure the validity of the responses.

(4) Survey more software professionals. Include a larger proportion of persons involved with software development and fewer persons from the academic and research areas. The intent is to get more data from people developing software as opposed to those in an academic environment.

A revised study based on a new methodology could involve a more detailed study of only a few metrics. One possible research effort could be performed using one or several metrics to measure a characteristic of an actual software requirements specification and comparing those measures to the perceptions software professionals have of that specific quality. For example, the IEEE measure of specification completeness could be used to measure the completeness of a specification. The results of that measurement could then be compared to the opinions software professionals have of the completeness of the specification.

## Appendix A

### Software Requirements Analysis Metrics Questionnaire

This questionnaire is designed to determine the types of software metrics which may be useful during the requirements analysis phase of software development. Metrics applicable to the requirements analysis process and to the software requirements specification (either formal or informal) are discussed.

For each metric, please provide the appropriate responses to the following six statements using the scale shown below:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.)

2. The data needed to calculate this metric is easily obtained before or during requirements analysis.

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it.

4. This metric measures the quality intended to be measured.

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable).

6. This metric would be useful during requirements analysis.

Please also provide any additional comments you may have directly on the questionnaire.

This questionnaire should take about 45 minutes to complete. When you have completed it, please return it in the envelope provided.

Thanks very much for your participation.

## Requirements Metrics Questionnaire

Please describe your past experience with software development:

1) Approximately how many years have you been involved in software engineering or management? (circle one)

| 5 years or less | between 5 and 10 years | 10 years or more |
|---|---|---|

2) Approximately how many times have you performed or been involved with software requirements analyses? (circle one)

| 2 times or less | between 2 and 5 times | between 5 and 10 times | 10 times or more |
|---|---|---|---|

3) Please provide your present title and briefly describe your experience with software development.

Present title: _____

Experience: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

## Schedule Progress

Definition: This metric measures the ability to maintain the software development schedule by tracking the delivery of software work packages defined in the work breakdown structure.

Primitives:

Program Schedule = number of months into development
BCWP = budgeted cost of work performed
BCWS = budgeted cost of work scheduled

Implementation:

$$\text{Estimated Schedule (months)} = \frac{\text{Program Schedule (months)}}{\text{BCWP/BCWS}}$$

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Requirements Documentation Progress

Definition: This measure compares the actual number of requirements documented with the estimated total number of requirements to be documented

Primitives:

Number of requirements specified
Estimated total number of requirements to be specified

Implementation:

$$\text{Progress} = \frac{\text{Number of requirements documented}}{\text{Estimated total number of requirements}} \times 100\%$$

NOTE: Number of requirements can also be replaced with any objective count such as functions, pages, etc.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Fault-Days Number

Definition: This measure represents the number of days that faults spend in the software system from creation to removal. The goal is to prevent reoccurrence of similar errors and to detect faults earlier.

Primitives:

(1) Phase when the fault was introduced in the system.
(2) Date when the fault was introduced in the system.
(3) Phase, date, and time when the fault is removed.

Implementation: For each fault detected and removed, the number of days from its creation to its removal is determined (fault-days = $FD_i$).

$FD_i$ = fault days for the $i^{th}$ fault

The fault-days are then summed for all faults detected and removed, to get the fault-days number at system level, including all faults detected/removed up to the delivery date. In cases when the creation date is not known, the fault is assumed to have been created at middle of the phase in which it was introduced.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Error Distribution Measure

Definition: This measure involves the analysis of the defect data collected during each phase of the software development and allows ranking of the predominant failure modes. The goal is to prevent reoccurrence of similar errors and to detect faults earlier.

Primitives:

(1) Fault type
(2) Fault severity
(3) Phase introduced
(4) Preventive measure
(5) Discovery mechanism

Implementation: The primitives for each error are recorded and the errors are counted according to criteria adopted for fault classification. The number of errors are then plotted for each class. The errors are classified and counted by phase, by cause, and by discovery mechanism.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Manhours Per Major Defect Detected

Definition: A measure of the manhours per defect detected during software product reviews and inspections. This measure provides a quantitative figure that is used to evaluate the efficiency of the review and inspection process.

Primitives:

$T_1$ = time expended by the inspection (or review) team in preparation for inspection (or review)

$T_2$ = time expended by the inspection (or review) team in conduct of inspection (or review)

$S_i$ = number of major (non-trivial) defects detected during the $i^{th}$ inspection (or review)

I = total number of inspections (or reviews) to date

Implementation: Record the preparation time (T1) and total time expended in conducting each meeting (T2). Defects are recorded and grouped into major/minor categories. (A major defect must be corrected for the product to function as desired.) Times are summarized and defects cumulatively added. The manhours per major defect detected is calculated:

$$M = \frac{\sum_{i=1}^{I} (T_1 + T_2)_i}{\sum_{i=1}^{I} S_i}$$

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

## Requirements Understandability

Definition: This is a measure of the understandability of semantic requirements specifications (i.e. written in English) using the Flesch-Kincaid readability formula.

Primitives: The Flesch-Kincaid formula has two factors:
(1) sentence length in words
(2) word length in syllables

Implementation: The measure of understandability is provided as a reading grade level (GL) according to the formula:

GL = 0.39 (Average number of words per sentence)
+ 11.8 (Average number of syllables per word)

The measure may be made of complete or partial requirements specifications, or of single requirement statements.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Requirement Ambiguity

Definition: This measure is used to estimate the ambiguity of a requirement, and to identify ambiguous requirements. This ambiguity poll is performed whenever a piece of requirements work is said to be finished.

Primitives: None.

Implementation: The poll is conducted as follows:

1. Gather a group of people to answer questions about the document whose ambiguity is to be measured. (The group should be as diverse as possible, at the very least including a sample from each population that will be affected by the eventual product.)

2. Be sure that there is no pressure to conform or no influence of any sort of one participant on another.

3. Propose a set of questions which can be answered with a number such as: How fast? How big? What capacity?

4. Estimate the ambiguity by comparing the highest and lowest answers.

5. Interview the high and low estimators to locate the sources of the ambiguity.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

## Requirements Traceability

Definition: This measure aids in identifying requirements that are either missing from, or in addition to, the original requirements.

Primitives:

R1 = number of requirements met by the architecture
R2 = number of original requirements

Implementation: A set of mappings from the requirements in the software architecture to the original requirements is created. Count each requirements met by the architecture (R1) and count each of the original requirements (R2). Compute the traceability measure (TM):

$$TM = \frac{R1}{R2} \times 100\%$$

When all of the original requirements are covered in the software architecture, the traceability measure is 100%. A measure of less than 100% indicates that some requirements have not been included in the software architecture.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Number of Conflicting Requirements

Definition: This measure is used to determine the reliability of a software system, resulting from the software architecture under consideration, as represented by a specification based on the entity-relationship-attribute model.

Primitives:

List of the inputs
List of the outputs
List of the functions performed by the program

Implementation: The mappings from the software architecture to the requirements are identified. Mappings from the same specification item to more than one differing requirement are examined for requirements inconsistency. (If the same specification item maps to two different requirements items, the requirements should be identical. Otherwise, the requirements are inconsistent.) Mappings from more than one specification item to a single requirement are examined for specification inconsistency. (If more than one specification item maps to a single requirement, the specification should be checked for possible inconsistency.)

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Requirements Compliance

Definition: This analysis is used to verify requirements compliance by using system verification diagrams (SVDs), a logical interconnection of stimulus response elements, which detect inconsistencies, incompleteness, and misinterpretations.

Primitives:

Decomposition elements (DEs):

- Stimulus – external input
- Function – defined input/output process
- Response – result of the function
- Label – numerical DE identifier
- Reference – specification paragraph number

Requirement errors detected using SVDs:

- $N_1$ = number due to inconsistencies
- $N_2$ = number due to incompleteness
- $N_3$ = number due to misinterpretation

Implementation: The implementation of an SVD is composed of the following phases:

(1) The decomposition phase is initiated by mapping the system requirement specifications into stimulus/response elements (DEs). That is, all keywords, phrases, functional and/or performance requirements and expected outputs are documented on decomposition forms.

(2) The graph phase uses the DEs from the decomposition phase and logically connects them to form the SVD graph.

(3) The analysis phase examines the SVD from the graph phase by using connectivity and reachability matrices. The various requirement error types are determined by examining the SVD and identifying errors as follows:

(a) Inconsistencies – Decomposition elements that do not accurately reflect the system requirement specification.

(b) Incompleteness – Decomposition elements that do not completely reflect the system requirement specification.

(c) Misinterpretation – Decomposition elements that do not correctly reflect the system requirement specification. These errors may occur during translation of the requirements into decomposition elements, constructing the SVD graph, or interpreting the connectivity and reachability matrices.

An analysis is also made of the percentages for the various requirements error types for the respective categories: inconsistencies, incompleteness, and misinterpretation.

$$\text{Inconsistencies (\%)} = \frac{N_1}{(N_1 + N_2 + N_3)} \times 100$$

$$\text{Incompleteness (\%)} = \frac{N_2}{(N_1 + N_2 + N_3)} \times 100$$

$$\text{Misinterpretation (\%)} = \frac{N_3}{(N_1 + N_2 + N_3)} \times 100$$

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Specification Completeness

Definition: This measure is used to determine the completeness of the software specification and to identify problem areas within the software specification.

Primitives:

$B_1$ = number of functions not satisfactorily defined
$B_2$ = number of functions
$B_3$ = number of data references not having an origin
$B_4$ = number of data references
$B_5$ = number of defined functions not used
$B_6$ = number of defined functions
$B_7$ = number of referenced functions not defined
$B_8$ = number of referenced functions
$B_9$ = number of decision points not using all conditions, options
$B_{10}$ = number of decision points
$B_{11}$ = number of condition options without processing
$B_{12}$ = number of condition options
$B_{13}$ = number of calling routines with parameters not agreeing with defined parameters
$B_{14}$ = number of calling routines
$B_{15}$ = number of condition options not set
$B_{16}$ = number of set condition options having no processing
$B_{17}$ = number of set condition options
$B_{18}$ = number of data references having no destination

Implementation: The completeness measure (CM) is the weighted sum of ten derivatives expressed as:

$$CM = \sum_{i=1}^{10} w_i D_i$$

where for each i=1,...,10, each weight $w_i$ has a value between 0 and 1, the sum of the weights is equal to 1, and each $D_i$ is a derivative with a value between 0 and 1.

To calculate the completeness measure: (1) The definitions of the primitives for the particular application must be determined, and (2) the priority associated with the derivatives must be determined. This prioritization affects the weights used to calculate the completeness measure.

Each derivative is determined as follows:

$$D_1 = \frac{(B_2 - B_1)}{B_2}$$ = functions satisfactorily defined

$D_2 = \frac{(B_4 - B_3)}{B_4}$ = data references having an origin

$D_3 = \frac{(B_6 - B_5)}{B_6}$ = defined functions used

$D_4 = \frac{(B_8 - B_7)}{B_8}$ = referenced functions defined

$D_5 = \frac{(B_{10} - B_9)}{B_{10}}$ = all condition options at decision points

$D_6 = \frac{(B_{12} - B_{11})}{B_{12}}$ = all condition options with processing at decision points used

$D_7 = \frac{(B_{14} - B_{13})}{B_{14}}$ = calling routine parameters that agree with the called routines defined parameters

$D_8 = \frac{(B_{12} - B_{15})}{B_{12}}$ = all condition options that are set

$D_9 = \frac{(B_{17} - B_{16})}{B_{17}}$ = processing follows set condition options

$D_{10} = \frac{(B_4 - B_{18})}{B_4}$ = data references that have a destination

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## Cause and Effect Graphing

Definition: Cause and effect graphing aids in identifying incomplete and ambiguous requirements. This measure explores the inputs and expected outputs of a program and identifies the ambiguities. Once these ambiguities are eliminated, the specifications are considered complete and consistent. (NOTE: A cause and effect graph is a formal transformation of a natural language specification (for example, English) into its input conditions and expected outputs. The graph depicts a combinatorial logic network.)

Primitives:

List of causes: distinct input conditions

List of effects: distinct output conditions or system transformation (effects are caused by the changes in the state of the system)

$A_{existing}$ = number of ambiguities in a program remaining to be eliminated

$A_{tot}$ = total number of ambiguities identified

Implementation: Identify all requirements and divide them into separate entities. Analyze the requirements to identify all the causes and effects in the specification. After the analysis is completed, assign each cause and effect a unique identifier. (For example, E1 for effect one.)

Next, create the cause and effect graph:

(1) Represent each cause and each effect by a node identified by its unique number.

(2) Interconnect the cause and effect nodes by analyzing the semantic content of the specification and transforming it into a Boolean graph. Each cause and effect can be in one of two states: true or false. Using Boolean logic, set the possible states of the causes and determine under what conditions each effect will be present.

(3) Annotate the graph with constraints describing combinations of causes and effects that are impossible because of semantic or environmental constraints.

(4) Identify as an ambiguity any cause that does not result in a corresponding effect, any effect that does not originate with a cause as a source, and any combination of causes and effects that are inconsistent with the requirement specification or impossible to achieve.

The measure of ambiguities present is computed as follows:

$$CE(\%) = 100 \times \left(1 - \frac{A_{existing}}{A_{tot}}\right)$$

When all of the causes and effects are represented in the graph and no ambiguities exist, the measure is 100%. A measure of less than 100% indicates some ambiguities still exist.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments:

## "Bang" – A Functionality Measure

Definition: This metric is a measure of the function of the software to be delivered as perceived by the user. It is a measure of the software functionality as stated in requirements specification. This measure is based on a formal specification model consisting of data flow diagrams, object (entity-relationship) diagrams, and state-transition diagrams.

Primitives: This measure's principal primitives are:

| | Partitioning vehicle | is used to partition | to produce the primitive |
|---|---|---|---|
| 1 | Function network | system requirement | functional primitives |
| 2 | Data dictionary | system data | data elements |
| 3 | Object diagram | retained data | objects |
| 4 | Object diagram | retained data | relationships |
| 5 | State diagram | control characteristic | states |
| 6 | State diagram | control characteristic | transitions |

Twelve essential counts of these primitives provide the basic metrics from which the measure of "Bang" is formulated:

FP = the count of functional primitives lying inside the man-machine boundary

FPM = the count of modified manual functional primitives (functions lying outside the man-machine boundary that must be changed to accommodate installation of the new automated system)

DE = the count of all data elements existing at and inside the man-machine boundary

DEI = the count of input data elements – those moving from manual primitives to automated primitives

DEO = the count of output data elements – those moving from automated to manual primitives

DER = the count of data elements retained (stored) in automated form

OB = the count of objects in the retained data model (automated portion only)

RE = the count of relationships in the retained data model (automated portion only)

ST = the count of states in the state transition model

TR = the count of transitions in the state transition model

$TC_i$ = the count of data tokens around the boundary of the $i^{th}$ functional primitive (evaluated for each primitive); a token is a data item that need not be subdivided within the primitive

$RE_i$ = the count of relationships involving the $i^{th}$ object of the retained data model (evaluated for each object)

Implementation: Measures of "Bang" are computed as follows:

Bang = FP x (weighting-factor-for-FP) +

DE x (weighting-factor-for-DE) + ...

A simpler and more productive way to characterize "Bang" is to choose one of the counts as a principal indicator and use the others to modify it. For most systems, FP is the principal indicator.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments: Specifically, does a function type measure such as this make for a good requirements analysis metric?

## Composite Specification Measures

Definition: Requirements are represented with a composite specification model (comprised of three views and corresponding notations; see table below), from which numerical counts are taken of various components of the representation.

| Composite Specification Model (CSM) | |
|---|---|
| Viewpoint | Notation |
| Functional | Data Flow |
| Contextual | Entity-Relationship |
| Dynamic | State-Transition |

The measures taken from the CSM are intended to capture the context of the system (hence the composite nature of the representation) in order to better understand it.

Primitives: This measure's primitives are organized according to the three views of the CSM.

Primitives associated with the functional view include: functions, interfaces, internal arcs, internal data items, system input/output data items, and file input/output data items.

Primitives associated with the contextual view include: entities, events, relationships, attributes, and value sets.

Primitives associated with the dynamic view include: states and transitions.

Implementation: Measures are also organized according to the three views of the CSM:

Measures associated with the functional view include:

Weighted function count

Numerical count of functional primitives

Numerical count of interfaces

Numerical count of internal arcs

Numerical count of internal data items

Numerical count of system input/output data items

Numerical count of file input/output data items

The measures associated with the contextual view include:

Numerical count of entities

Numerical count of events

Numerical count of relationships

Numerical count of attributes

Numerical count of value sets]

The measures associated with the dynamic view include:

Numerical count of states

Numerical count of transitions

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments: Specifically, are these types of measures good requirements analysis metrics?

## Control Flow Measures

Definition: The following measurements were originally created for requirements written in the control flow requirement specification language RSL (based on the control flow and entity–relationship models). The control flow model is implemented through "The Requirement Network (R_NET)"; a control flow graph consisting of nodes (specifying processing operations) and connecting arcs. The measures are intended to measure the complexity of the specified system in order to better understand and manage the system development. The measures are also intended to measure other functional and non-functional attributes of the requirements specification, as identified below, in order to enhance the quality of the requirements specification.

Primitives: Each measure has its own primitive, which are identified in the definition of each measure.

Implementation: The measures are simple counts of the following items, except where specified otherwise.

To infer system complexity:

Number of requirement networks (R_NETs)

Number of processing activities (ALPHAs)

Number of requirement networks (R_NETs) that are enabled directly or indirectly through a sequence of other R_NETs

Number of global variables

Number of requirement networks (R_NETs) and processing activities (ALPHAs) that read or write global variables

Number of requirement networks (R_NETs) and processing activities (ALPHAs) that must be changed if a certain data structure is modified

To infer correctness of the requirements:

Number of functions (number of R_NETs and ALPHAs in RSL)

Number of states in STM

To infer understandability of the requirements:

Nesting level of OR–nodes (predicate nodes) in a piece of software

Number of states in STM

Number of data items

Usage of global data versus local data

Degree of data abstraction

Degree of data dependency

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments: Specifically, do control flow measures make for good requirements analysis metrics?

## Miscellaneous Explicit Counts

Definition: The following measures are simple explicit counts of various requirements specification attributes. The counts are provided to allow you, the survey participant, an opportunity to comment on various simple measures of the requirements specification product and process.

Primitives: Each count has its own primitive. The primitives and measures are self-evident.

Implementation: Simple counts of the following requirements specification attributes are made in order to measure the quality specified.

Requirements completeness:

Number of TBDs in the specification

Number of non-existent references

Number of missing specification items

Number of missing functions

Number of missing products.

Requirements consistency:

Number of conflicting requirements

Number of non-traceable requirements.

Requirements testability:

Number of testable requirements

Number of untestable requirements.

Number of unique requirements

Number of TBDs

Requirements specification process effectiveness:

Number of defects found during reviews

Number of problem reports generated

Number of change requests generated

Number of completed change orders

Number of open change requests

System size and functionality:

Number of pages in the requirements specification

Number of input/output requirements

Number of constraints

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments: Specifically, do simple measures make good metrics?

## Completeness Checklist

Definition: This checklist measures the completeness of the requirements specification.

Primitives: As shown in the checklist below.

Implementation: The requirements specification is inspected using the following checklist to determine the completeness of the specification.

| Completeness Checklist | | |
|---|---|---|
| # | Item | Score |
| 1 | Unambiguous References<br>Are requirements itemized so various functions and their inputs/outputs are clearly delineated?<br>YES = 1 NO = 0 | |
| 2 | External Data References<br>2.1 Number of data references which are defined.<br>2.2 Number of major data references.<br>SCORE = 2.1 + 2.2 | |
| 3 | Major Functions Used<br>3.1 Number of defined functions used.<br>3.2 Number of functions identified.<br>SCORE = 3.1 + 3.2 | |
| 4 | Major Functions Defined<br>4.1 Number of identified functions defined.<br>4.2 Number of functions identified.<br>SCORE = 4.1 + 4.2 | |
| 5 | Decision Points Defined<br>Is the flow of processing and all decision points in that flow defined?<br>YES = 1 NO = 0 | |

The scores are then compared to organizational or individual project requirements or goals which are generally based on historical data.

(Questions are on next page.)

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments: Specifically, do checklist type measures make good metrics?

## Requirements Analysis Worksheet

Definition: This worksheet contains measurements (in the form of numerous questions) for various quality factors to be applied to the requirements specification.

Primitives: None.

Implementation: The requirements specification is inspected using the following worksheet to determine the relative quality of the specification. (NOTE: For brevity, only a few of the many quality factor worksheet items are provided.)

| Requirements Analysis Worksheet | | | |
|---|---|---|---|
| # | Topic and Item | Answer Yes | No |
| 1.0 | Structure | | |
| 1.1 | Is an organization of the system provided which identifies all functions and interfaces in the system? | | |
| 1.2 | Are there duplicate functions and/or interfaces? | | |
| 1.3 | Is there a definitive statement of the requirements for the distribution of information within the database? | | |
| 1.4 | Is an organization of the database provided which identifies the types of system-level information and the information flow within the system? | | |
| 1.5 | Is there a definitive statement of requirements for code to be written according to a coding standard? | | |
| 1.6 | Is there a definitive statement of requirements for processes, functions, and modules to have loose coupling and high cohesion? | | |
| 2.0 | Completeness and Correctness | | |
| 2.1 | Is there a matrix relating itemized requirements to major functions which implement those requirements? | | |
| 2.1 | Are requirements itemized so that various functions, their inputs and outputs, are clearly delineated? | | |
| 2.3 | Is the flow of processing and all decision points in that flow described? | | |
| 2.4 | Are all functions identified clearly defined? | | |
| 2.5 | Are all data references identified clearly defined? | | |
| 2.8 | Are all defined functions used? | | |
| 2.9 | Are all defined data references used? | | |

The answers are then compared to organizational or individual project requirements or goals which are generally based on historical data.

Please use this scale to respond to the following statements:

| (a) Strongly Agree | (b) Agree | (c) Neither Agree Nor Disagree | (d) Disagree | (e) Strongly Disagree |
|---|---|---|---|---|

1. This metric is simple to understand and precisely defined. (i.e., It is clear how this metric is evaluated.) _____

2. The data needed to calculate this metric is easily obtained before or during requirements analysis. _____

3. The benefits derived from using this metric outweigh the costs and effort of obtaining the data to use it. _____

4. This metric measures the quality intended to be measured. _____

5. This metric is insensitive to small changes in the requirements analysis process or product (as applicable). _____

6. This metric would be useful during requirements analysis. _____

Comments: Specifically, are worksheets better measures of requirements specification quality than more typical metrics?

## Appendix B

## Interview Discussion Topics

1. Metrics participant perceived as having utility include:

_____

_____

2. Metrics participant perceived as not having utility include: _____

_____

_____

3. Perceptions the participant has of a useful requirement metric include: _____

_____

_____

_____

4. Rationale the participant used to justify rating metrics as not having utility: _____

_____

_____

_____

5. Additional comments: _____

_____

_____

_____

# Appendix C

## Questionnaire Response Data

### Mean Scores and Overall Scores

| | Mean Score on Statement Number | | | | | | Overall |
|---|---|---|---|---|---|---|---|
| Matric Title | St 1 | St 2 | St 3 | St 4 | St 5 | St 6 | Score |
| Schedule Progress | 3.7 | 3.0 | 3.2 | 2.5 | 3.3 | 2.8 | 21.3 |
| Rqmts Doc Progress | 3.5 | 2.8 | 3.2 | 3.0 | 3.2 | 3.3 | 22.3 |
| Fault Days Number | 3.0 | 2.3 | 3.0 | 3.2 | 3.0 | 2.8 | 20.2 |
| Error Dist Measure | 3.2 | 2.8 | 3.7 | 3.8 | 3.5 | 3.2 | 23.3 |
| Manhours per Defect | 3.2 | 3.0 | 3.5 | 3.0 | 3.3 | 3.5 | 23.0 |
| Rqmts Understand | 4.0 | 3.2 | 2.2 | 2.3 | 2.7 | 1.8 | 18.0 |
| Rqmts Ambiguity | 1.8 | 1.7 | 1.7 | 2.2 | 3.0 | 2.3 | 15.0 |
| Rqmts Traceability | 3.3 | 1.5 | 3.2 | 3.2 | 3.5 | 3.2 | 21.0 |
| No. Conflict Rqmt | 2.5 | 2.0 | 2.5 | 2.5 | 2.7 | 2.5 | 17.2 |
| Rqmts Compliance | 2.5 | 2.7 | 2.7 | 3.2 | 3.3 | 3.3 | 21.0 |
| Spec Completeness | 2.7 | 2.7 | 2.8 | 3.2 | 3.2 | 3.2 | 20.8 |
| Cause Effect Graph | 2.7 | 1.8 | 2.5 | 3.3 | 3.2 | 3.3 | 20.2 |
| Bang Measure | 2.0 | 2.5 | 2.5 | 2.8 | 3.3 | 2.8 | 18.8 |
| CSM Measures | 1.5 | 2.2 | 2.7 | 3.2 | 3.5 | 2.7 | 18.3 |
| Control Flow Meas | 2.5 | 2.5 | 3.2 | 3.2 | 3.5 | 3.5 | 21.8 |
| Misc Counts | 3.3 | 2.8 | 3.5 | 3.3 | 3.2 | 3.5 | 23.2 |
| Checklist | 3.0 | 3.2 | 3.3 | 3.0 | 2.8 | 3.5 | 22.3 |
| Worksheet | 3.5 | 3.0 | 2.8 | 2.7 | 3.2 | 3.7 | 22.5 |

NOTES:

Lowest possible mean score = 1

Highest possible mean score = 5

Lowest possible overall score = 7

Highest possible overall score = 35

## Maximum Overall Scores, Minimum Overall Scores, and Overall Scores

| Metric Title | Maximum Overall Score | Minimum Overall Score | Overall Score |
|---|---|---|---|
| Schedule Progress | 27 | 15 | 21.3 |
| Rqmts Doc Progress | 29 | 13 | 22.3 |
| Fault Days Number | 28 | 7 | 20.2 |
| Error Dist Measure | 33 | 17 | 23.3 |
| Manhours per Defect | 34 | 19 | 23.0 |
| Rqmts Understand | 23 | 11 | 18.0 |
| Rqmts Ambiguity | 19 | 10 | 15.0 |
| Rqmts Traceability | 30 | 13 | 21.0 |
| No. Conflict Rqmt | 27 | 7 | 17.2 |
| Rqmts Compliance | 27 | 15 | 21.0 |
| Spec Completeness | 26 | 11 | 20.8 |
| Cause Effect Graph | 26 | 14 | 20.2 |
| Bang Measure | 26 | 11 | 18.8 |
| CSM Measures | 26 | 11 | 18.3 |
| Control Flow Meas | 28 | 13 | 21.8 |
| Misc Counts | 29 | 16 | 23.2 |
| Checklist | 29 | 7 | 22.3 |
| Worksheet | 30 | 14 | 22.5 |

## Overall Scores and Standard Deviation for each Overall Score

| Metric Title | Overall Score + StdDev | Overall Score - StdDev | Overall Score | St Dev |
|---|---|---|---|---|
| Schedule Progress | 25.9 | 16.7 | 21.3 | 4.59 |
| Rqmts Doc Progress | 28.5 | 16.2 | 22.3 | 6.12 |
| Fault Days Number | 28.0 | 12.3 | 20.2 | 7.83 |
| Error Dist Measure | 30.6 | 16.0 | 23.3 | 7.28 |
| Manhours per Defect | 29.0 | 17.0 | 23.0 | 6.03 |
| Rqmts Understand | 22.0 | 14.0 | 18.0 | 4.00 |
| Rqmts Ambiguity | 18.3 | 11.7 | 15.0 | 3.35 |
| Rqmts Traceability | 27.4 | 14.6 | 21.0 | 6.42 |
| No. Conflict Rqmt | 24.7 | 9.6 | 17.2 | 7.55 |
| Rqmts Compliance | 25.4 | 16.6 | 21.0 | 4.38 |
| Spec Completeness | 26.7 | 14.9 | 20.8 | 5.91 |
| Cause Effect Graph | 25.1 | 15.2 | 20.2 | 4.96 |
| Bang Measure | 25.4 | 12.2 | 18.8 | 6.59 |
| CSM Measures | 23.6 | 13.1 | 18.3 | 5.28 |
| Control Flow Meas | 27.5 | 16.1 | 21.8 | 5.71 |
| Misc Counts | 28.2 | 18.1 | 23.2 | 5.08 |
| Checklist | 30.3 | 14.4 | 22.3 | 7.97 |
| Worksheet | 28.6 | 16.4 | 22.5 | 6.09 |

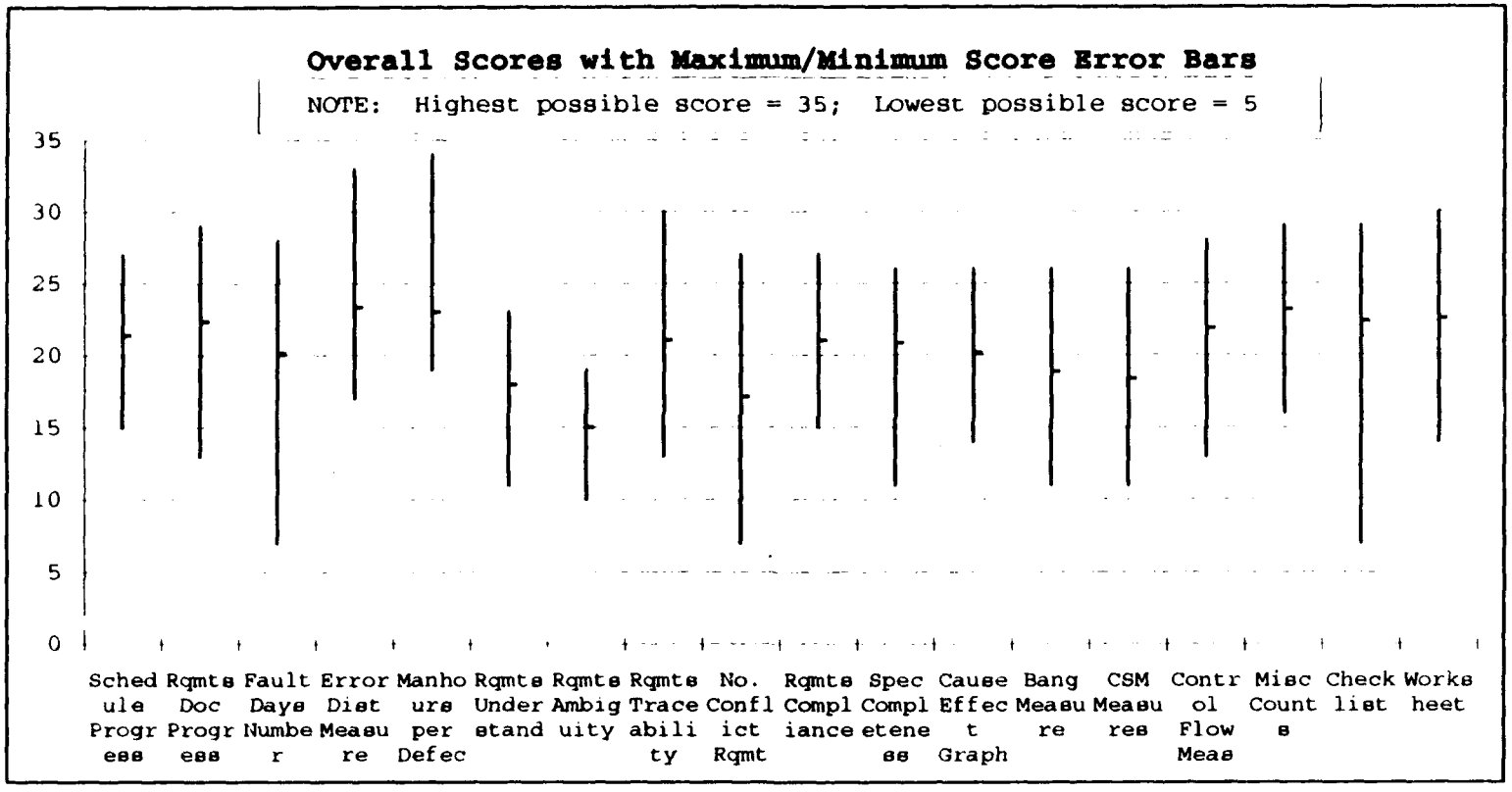
## Maximum Scores, Minimum Scores, and Mean Scores for Statement Number Six

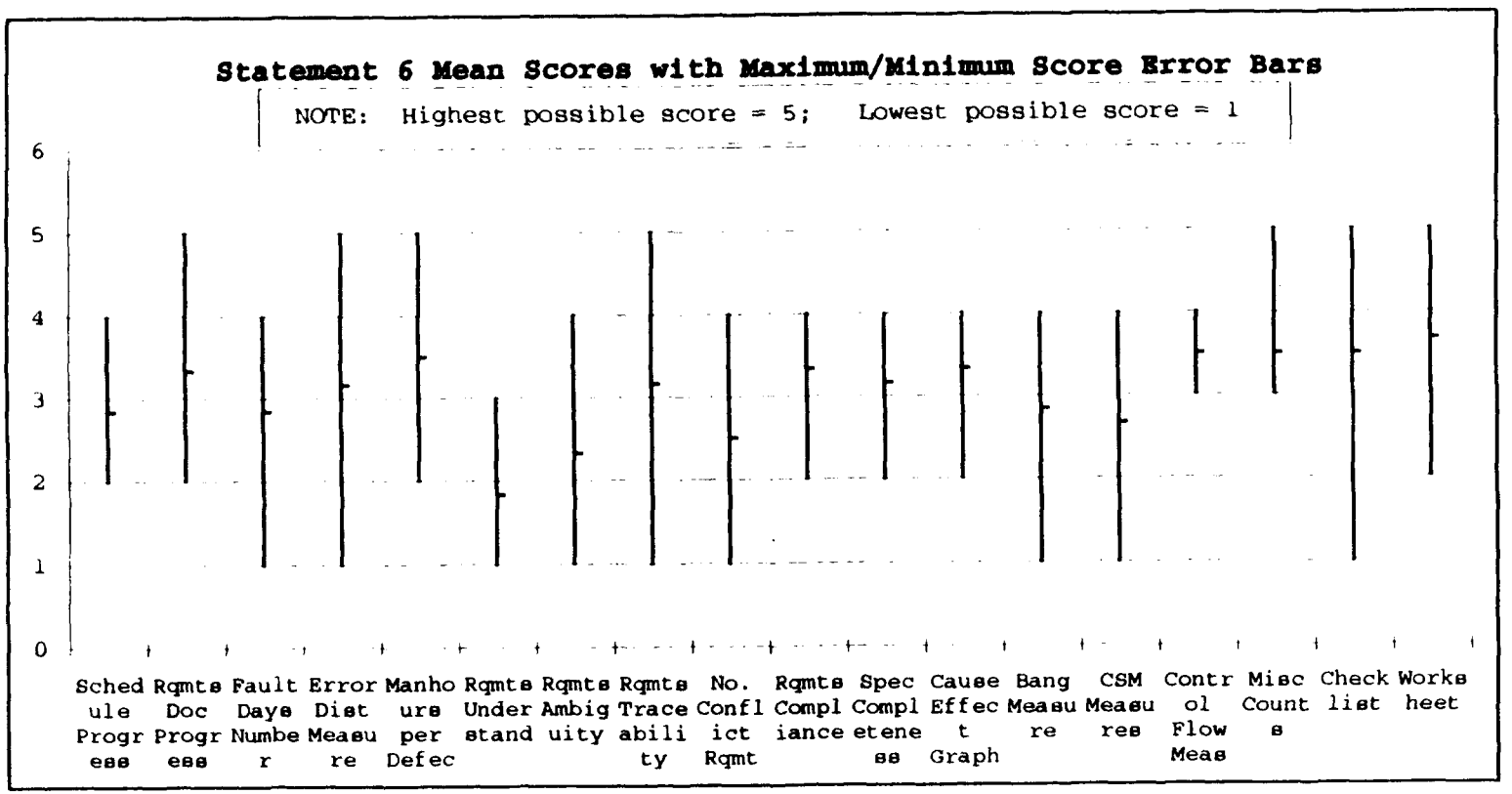
| Metric Title | Maximum Score Statement 6 | Minimum Score Statement 6 | Mean Score Statement 6 |
|---|---|---|---|
| Schedule Progress | 4 | 2 | 2.8 |
| Rqmts Doc Progress | 5 | 2 | 3.3 |
| Fault Days Number | 4 | 1 | 2.8 |
| Error Dist Measure | 5 | 1 | 3.2 |
| Manhours per Defect | 5 | 2 | 3.5 |
| Rqmts Understand | 3 | 1 | 1.8 |
| Rqmts Ambiguity | 4 | 1 | 2.3 |
| Rqmts Traceability | 5 | 1 | 3.2 |
| No. Conflict Rqmt | 4 | 1 | 2.5 |
| Rqmts Compliance | 4 | 2 | 3.3 |
| Spec Completeness | 4 | 2 | 3.2 |
| Cause Effect Graph | 4 | 2 | 3.3 |
| Bang Measure | 4 | 1 | 2.8 |
| CSM Measures | 4 | 1 | 2.7 |
| Control Flow Meas | 4 | 3 | 3.5 |
| Misc Counts | 5 | 3 | 3.5 |
| Checklist | 5 | 1 | 3.5 |
| Worksheet | 5 | 2 | 3.7 |

## Mean Scores and Standard Deviation for

## Statement Number Six

| Matric Title | Mean Score + StdDev | Mean Score - StdDev | Mean Score | St Dev |
|---|---|---|---|---|
| Schedule Progress | 3.8 | 1.9 | 2.8 | 1.0 |
| Rqmts Doc Progress | 4.4 | 2.3 | 3.3 | 1.0 |
| Fault Days Number | 4.3 | 1.4 | 2.8 | 1.5 |
| Error Dist Measure | 4.8 | 1.6 | 3.2 | 1.6 |
| Manhours per Defect | 4.5 | 2.5 | 3.5 | 1.0 |
| Rqmts Understand | 2.6 | 1.1 | 1.8 | 0.8 |
| Rqmts Ambiguity | 3.5 | 1.1 | 2.3 | 1.2 |
| Rqmts Traceability | 4.6 | 1.7 | 3.2 | 1.5 |
| No. Conflict Rqmt | 3.9 | 1.1 | 2.5 | 1.4 |
| Rqmts Compliance | 4.1 | 2.5 | 3.3 | 0.8 |
| Spec Completeness | 3.9 | 2.4 | 3.2 | 0.8 |
| Cause Effect Graph | 4.1 | 2.5 | 3.3 | 0.8 |
| Bang Measure | 4.0 | 1.7 | 2.8 | 1.2 |
| CSM Measures | 3.7 | 1.6 | 2.7 | 1.0 |
| Control Flow Meas | 4.0 | 3.0 | 3.5 | 0.5 |
| Misc Counts | 4.3 | 2.7 | 3.5 | 0.8 |
| Checklist | 4.9 | 2.1 | 3.5 | 1.4 |
| Worksheet | 4.9 | 2.5 | 3.7 | 1.2 |


Overall Scores with Maximum/Minimum Score Error Bars
NOTE: Highest possible score = 35; Lowest possible score = 5
35
30
25
20
15
10
5
0
Schedule Progress
Rqmts Doc Progress
Fault Days Number
Error Dist Measure
Manhours per Defec
Rqmts Understand
Rqmts Ambiguity
Rqmts Traceability
No. Conflict Rqmt
Rqmts Compliance
Spec Completeness
Cause Effect Graph
Bang Measure
CSM Measures
Control Flow Meas
Misc Counts
Checklist
Worksheet


Statement 6 Mean Scores with Maximum/Minimum Score Error Bars
NOTE: Highest possible score = 5; Lowest possible score = 1
6
5
4
3
2
1
0
Schedule Progress
Rqmts Doc Progress
Fault Days Number
Error Dist Measure
Manhours per Defec
Rqmts Understand
Rqmts Ambiguity
Rqmts Traceability
No. Conflict Rqmt
Rqmts Compliance
Spec Completeness
Cause Effect Graph
Bang Measure
CSM Measures
Control Flow Meas
Misc Counts
Check list
Worksheet

## Bibliography


Agresti, William W. "An Approach to Developing Specification Measures," Proceedings, Ninth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, (November 1984).

Basili, Victor R. and H. Dieter Rombach. "Tailoring the Software Process to Project Goals and Environments," Proceedings of the Ninth International Conference on Software Engineering. 345-357. IEEE Computer Society Press, (30 March-2 April 1987).

Boehm, Barry. "Verifying and Validating Software Requirements and Design Specifications," IEEE Software, 1:75-88 (January 1984).

Boeing Aerospace Company. Software Quality Measurement for Distributed Systems, Volumes 1-3. Air Force Systems Command Rome Air Development Center Technical Reports RADC-TR-83-175-VOL-1, RADC-TR-83-175-VOL-2, RADC-TR-83-175-VOL-3, July 1983 (AD-A137 955, AD-A137 956, AD-A137 957).

Bush, Marilyn W. Software Product Assurance Metrics Study: JPL's Software Systems Quality and Productivity. JPL Publication 89-6. Pasadena CA: NASA Jet Propulsion Laboratory, 15 February 1989.

Cioch, Frank A. "Measuring Software Misinterpretation," The Journal of Systems and Software, 15:85-95 (February 1991).

Conte, S. D. and others. Software Engineering Metrics and Models. Menlo Park CA: Benjamin/Cummings Publishing Company, Inc., 1986.

DeMarco, Tom. Controlling Software Projects. Englewood Cliffs NJ: Yourdon Press, 1982.

Department of the Air Force. Acquisition Management: Software Management Indicators. AFSC Pamphlet 800-43. Andrews AFB DC: HQ AFSC, 31 August 1990.

Dziegiel, Roger J., Jr. C2 Software Engineering Branch, Air Force Systems Command Rome Laboratories, Griffiss AFB NY. Electronic mail message. 2 April 1991.

Eisenberg, Anne. Effective Technical Communication. New York: McGraw-Hill, Inc., 1982.

Emory, C. William. Business Research Methods (Revised Edition). Homewood IL: Richard D. Irwin, Inc., 1980.

Farbey, B. "Software Quality Metrics: Considerations About Requirements and Requirement Specifications," Information and Software Technology, 32:60–64 (January/February 1990).

Fouser, Thomas J. Software Requirements Analysis Phase Trial Standard. JPL D–4005 (Version 3.0). Pasadena CA: NASA Jet Propulsion Laboratory, December 1988.

Gause, Donald C. and Gerald M. Weinberg. Exploring Requirements. Quality Before Design. New York: Dorset House Publishing Company, Inc., 1989.

IEEE Guide to Software Requirements Specifications. ANSI/IEEE Std 830–1984, 20 July 1984.

IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE Std 982.2–1988 (Corrected Edition), 12 June 1989.

IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE Std 982.1–1988, 9 June 1988.

Kitchenham, Barbara A. and others. "An Evaluation of Some Design Metrics," Software Engineering Journal, 5:50–58 (January 1990).

Losa, J. W., CDR, USNR, and others. Readability Grade Levels of Selected Navy Technical School Curricula. USN Training Analysis and Evaluation Group, Orlando FL, Technical Memorandum 83-2, February 1983 (AD–A125 862).

Mills, Everald E. Software Metrics (SEI Curriculum Module SEI–CM–12–1.1). Carnegie Mellon University Software Engineering Institute, December 1988.

National Research Council Air Force Studies Board. Adapting Software Development Policies to Modern Technology. Washington DC: National Academy Press, July 1989 (AD–A213 391/6).

Ramamoorthy, C. V. and others. "Software Quality and Requirement Specification," Proceedings IEEE Computer Society 1986 International Conference on Computer Languages. 75–83. IEEE Computer Society Press, New York, 1986.

-----. "Metrics Guided Methodology," Proceedings IEEE Computer Society's Ninth International Computer Software and Applications Conference (COMPSAC86). 111–120. IEEE Computer Society Press, New York, 1985.

Riney, Larry A. Technical Writing for Industry: An Operations Manual for the Technical Writer. Englewood Cliffs NJ: Prentice-Hall, Inc., 1989.

Ross, Niall. "Using Metrics in Quality Management," IEEE Software, 7:80–85 (July 1990).

Schultz, Herman P. Software Management Metrics. The MITRE Corporation, Bedford MA, May 1988 (AD–A196 916).

Shepperd, M. "Early Life-Cycle Metrics and Software Quality Models," Information and Software Technology, 32:311–316 (May 1990).

Systems Architects, Inc. Computer Systems Acquisition Metrics Handbook, Volumes 1–4. Air Force Systems Command Electronic Systems Division Technical Reports ESD–TR–82–143(1), ESD–TR–82–143(2), ESD–TR–82–143(3), ESD–TR–82–143(4), May 1982 (AD–A120 375, AD–A120 376, AD–A120 377, AD–A120 378).

## Vita

Captain James H. Byers was born on August 13, 1963 in Rockville Centre, New York. He graduated from General Douglas MacArthur High School in Levittown, New York in 1981 and attended the State University of New York at Buffalo, graduating with a Bachelor of Science Degree in Aerospace Engineering in May 1985. Captain Byers attended the Air Force Officer Training School in San Antonio, Texas during the summer of 1985 and, upon graduation, was commissioned a Second Lieutenant in the United States Air Force. He was then assigned to the 6595th Shuttle Test Group (Air Force Systems Command) at Vandenberg AFB, California, where he served initially as a Space Shuttle Systems Engineer and later as a Project Engineer. Following the Space Shuttle Challenger accident and the subsequent deactivation of the 6595th Shuttle Test Group, he served as a Project Engineer for the Western Space and Missile Center Titan IV/Centaur Launch Complex Program. He entered the Air Force Institute of Technology School of Systems and Logistics in May 1990.

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | December 1991 | Master's Thesis |

**4. TITLE AND SUBTITLE**
RELATIVE UTILITY OF SELECTED SOFTWARE REQUIREMENT METRICS

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
James H. Byers, Captain, USAF

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Air Force Institute of Technology, WPAFB OH 45433-6583

**8. PERFORMING ORGANIZATION REPORT NUMBER**
AFIT/GSS/LSY/91D-4

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

The objective of this study was to determine the relative utility of selected software requirement metrics in assessing the productivity of the software requirements analysis process and the quality of the products of this process. This objective was met by collecting information about the perceptions that practicing software professionals have of the usefulness of various requirement metrics. The study employed a two part methodology. The first part utilized Basili's goal/question/metric paradigm to identify specific goals of the measurement effort and to identify requirement metrics worthy of further investigation. The second part employed a typical research design to gather perceptions that software professionals have of the utility of several metrics selected from those identified earlier. The study produced inconclusive results and further research is recommended. Results were based on a small sample and the data only reiterated the mixed opinions that software professionals have of the usefulness of software metrics. One significant finding is the consensus that a metric must be precisely defined for it to be accepted by the software community.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Computer Programs, Software, Software Engineering, Requirements, Specifications, Measurement | 122 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

AFIT Control Number AFIT/GSS/LSY/91D-4

# AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to determine the potential for current and future applications of AFIT thesis research. Please return completed questionnaires to: AFIT/LSC, Wright-Patterson AFB OH 45433-6583.

1. Did this research contribute to a current research project?

   a. Yes  b. No

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not researched it?

   a. Yes  b. No

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency received by virtue of AFIT performing the research. Please estimate what this research would have cost in terms of manpower and/or dollars if it had been accomplished under contract or if it had been done in-house.

   Man Years _____ _____ $_____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3 above), what is your estimate of its significance?

   a. Highly Significant  b. Significant  c. Slightly Significant  d. Of No Significance

5. Comments

_____ _____

Name and Grade  Organization

_____ _____

Position or Title  Address